Stochastic Local Search Methods for Highly Constrained Combinatorial Optimisation Problems

Graph Colouring, Generalisations, and Applications

Dissertationsschrift

zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)

von
Dipl.-Ing. Marco Chiarandini
aus Udine, Italien

Referent: Prof. Dr. Wolfgang Bibel
Korreferent: Dr. habil. Thomas Stützle

Tag der Einreichung: 6. Mai 2005
Tag der mündlichen Prüfung: 8. Juli 2005

Genehmigte Dissertation vom Fachbereich Informatik
Darmstadt 2005
Hochschulkennziffer D17

This thesis is about advances in the application of Stochastic Local Search methods for solving graph colouring problems and highly constrained combinatorial optimisation problems that arise from real world systems.

Zusammenfassung

Das Graphenfärbeproblem besteht darin, die Knoten eines Graphen so zu färben, dass keine zwei durch eine Kante verbundenen Knoten die gleiche Farbe erhalten. Zusammen mit Verallgemeinerungen dieser Problemstellung taucht es als Kern vieler Probleme des täglichen Lebens wie der Frequenzzuweisung in Mobilfunknetzen oder bei der Erstellung eines Stundenplans für Vorlesungen an einer Universität auf. Ihre einfache Formulierung als Graphenfärbungsprobleme erlaubt eine systematische Untersuchung durch Reduktion auf den harten Kern dieser Probleme. All diese Probleme sind kombinatorische Optimierungsprobleme, die durch eine Reihe von Bedingungen an Lösungen und durch ein Optimierungskriterium charakterisiert werden.

Die Lösung komplexer Graphenfärbeprobleme kann in effizienter Weise durch Methoden der stochastisch lokalen Suche (SLS) erfolgen. Abstrakt gesehen bestehen viele SLS-Methoden aus mehreren Komponenten, nämlich einem Konstruktionsalgorithmus, einem iterativen Verbesserungsalgorithmus und einer Metakomponente, genannt Metaheuristik, die die beiden ersteren Komponenten steuert. Diese ersten beiden Komponenten sind stark problemabhängig und erfordern das Ausnutzen problemspezifischer Kenntnisse, während die Metaheuristik allgemeiner gehalten ist. Konkrete SLS-Algorithmen enstehen durch die Kombination verschiedener konkreter Komponenten, die wiederum jeweils weiter parametrisiert werden können. Die Konfiguration konkreter SLS-Algorithmen als Auswahl konkreter Komponenten und deren Parametrisierung ist eine komplexe Aufgabe, weil viele verschiedene Variationen konkreter Komponenten mit vielen Parametern und dementsprechend viele Kombinationen denkbar sind. Die Konfigurationsaufgabe muß auf empirischen Tests beruhen, da theoretische Erkenntnisse in diesem Kontext schwer und nur grob zu erhalten sind. Der ganze Prozeß von Design, Entwicklung und Konfiguration von SLS-Methoden und Algorithmen kann tatsächlich als ein Ingenieursprozess mit dem Ziel der systematischen Implementierung von SLS-Algorithmen aufgefasst werden.

Der Ausgangspunkt dieser Arbeit ist die Definition statistischer Verfahren für die Analyse von SLS-Algorithmen und allgemeiner von beliebigen stochastischen Optimierungsalgorithmen. Es wird gezeigt, dass die Annahmen bei der Anwendung parametrischer statistischer Tests oft verletzt sind, und dass deshalb oft zwei alternative Methoden, sogenannte Permutationstests und rangbasierte Tests, verwendet werden müssen. Im Rahmen dieser Dissertation werden Permutationstests weiterentwickelt und als weitere Möglichkeit zur Analyse von stochastischen Optimierungsalgorithmen eingeführt. Darüberhinaus wird aus der parametrischen Statistik eine graphische Darstellung der Resultate durch simultane Konfidenzintervalle übernommen und hier für nichtparametrische Testverfahren adaptiert. Der Vorteil dieser graphischen Darstellung ist die Vereinigung von Informationen der beschreibenden mit denen der schließenden Statistik in einer einzigen Graphik. Die entwickelten statistischen Methoden werden beispielhaft zur Analyse von SLS–Algorithmen für das Graphenfärbungsproblem und das Mengen–T-Färbungsproblem, einer wichtigen Verallgemeinerung des Graphenfärbungsproblems, angewendet.

Verschiedene SLS-Algorithmen sind in der Literatur zur Lösung des Graphenfärbungsproblems vorgeschlagen worden, aber ein unvoreingenommener, systematischer Vergleich zwischen diesen wurde bisher nicht unternommen. Eine ähnliche Situation gilt für das Mengen-T-Färbungsproblem. In beiden Fällen werden im Rahmen dieser Dissertation sowohl die bekanntesten Algorithmen reimplementiert als auch neue Algorithmen entwickelt und anschließend mittels eines strikten experimentellen Designs verglichen. Dadurch können Hinweise darauf erhalten werden, welches die besten Lösungsansätze zur Lösung der zwei betrachteten Probleme sind und welches der beste Algorithmus in Abhängigkeit von bestimmen Charakteristiken der konkreten Instanzen ist.

In einem letzten Schritt wird untersucht, wie verschiedene allgemeine SLS-Methoden zur Lösung einer Universitätsvorlesungsplanung (engl. course timetabling) angewendet und kombiniert werden können. Das Design von Komponenten für abgeleitete Algorithmen beruht einerseits auf Einsichten, die für die beiden Färbungsprobleme gewonnen wurden, und andererseits auf Anforderungen eines Algorithmenwettbewerbs, in dessen Rahmen die Algorithmen entwickelt wurden. Aus diesem Grunde wird ein spezieller Focus auf die systematische Entwicklung *eines* Algorithmus gelegt. Die Entwicklung von Algorithmuskomponenten und die Kombination zu einem kontreten SLS–Algorithmus wird als ein ingenieurmäßiger Prozess dargestellt, der aus der Wechselwirkung von Algorithmusdesign und experimentellen Tests besteht. Dieses Verfahren wird als angemessen für die Anwendung von beliebigen SLS-Methoden auf komplexe Probleme aus der realen Welt erachtet und begründet.

Die Hauptergebnisse dieser Dissertation sind die folgenden:

1. Beim Graphenfärbungsproblem bleibt die einfache Tabu–Suche mit einer Einer-austauschnachbarschaft ein sehr konkurrenzfähiger Ansatz; die Anwendung einer sehr großen Nachbarschaft ist nicht nutzbringend.

2. Beim Mengen–T–Färbungsproblem gibt es zwei gute Algorithmen, die auf der Einer-austauschnachbarschaft beruhen und jeweils auf einzelnen Instanzklassen die besten Algorithmen sind: ein adaptiver, iterativer greedy-Algorithmus für Graphen und ein Tabu–Suchalgorithmus, der durch eine eingeschränkte, exakte Zuweisung von Farben an Knoten erweitert wurde. Diese zwei Algorithmen können auch kombiniert werden.

3. Der Einsatz eines ingenieurmäßigen Prozesses zur Entwicklung von Algorithmen, der auf sequentiellem Tests beruhen, ist besonders geeignet für die erfolgreiche Anwendung von SLS-Methoden. Der Einsatz eines solchen Prozesses hat die Entwicklung eines Algorithmus für die Universitätsvorlesungsplanung erleichtert, der bei einem internationalen Wettbewerb unter 24 eingereichten Lösungen der beste war.

Summary

Graph colouring is a combinatorial optimisation problem consisting in colouring the vertices of a graph such that no vertices connected by an edge receive the same colour. The minimal number of colours for which such a colouring exists is an intrinsic property of the graph and is called chromatic number. Many real life situations, such as the frequency assignment in mobile networks or the scheduling of courses at a university, can be modelled in this way. Colouring planar graphs, such as maps can be easy, and four colours suffice, but real life systems are much more complex. When modelled by graph colouring, they entail general graphs of large size and include more sophisticated constraints than those representable by simple unweighted edges.

Stochastic Local Search (SLS) methods are approximate techniques for efficiently solving complex combinatorial optimisation problems. They typically consist of construction algorithms, iterative improvement algorithms, and meta-components, better known as metaheuristics. The first two are strongly problem dependent and require the exploitation of problem-specific knowledge, while the last are more general concepts to guide the first two components. The instantiation of SLS algorithms arises from the combination of concrete algorithmic components. The task of combining these concrete components in an effective algorithm is complex due to their many possible combinations and the need of determining a certain number of parameters. This task must necessarily rely on empirical tests and the whole process of designing, developing, and configuring an SLS algorithm is actually an engineering process.

The starting point of this work is the definition of the statistical methods that are appropriate for the analysis of stochastic algorithms for optimisation. We argue that the assumptions for the application of parametric statistical tests are often violated and opt for two alternative methods: permutation and rank-based tests. Our work contributes to the development of permutation tests and to their introduction into the analysis of algorithms for optimisation. Moreover, we transfer a graphical representation of results through simultaneous confidence intervals from the parametric to the non-parametric cases. This representation has the advantage of conveying in one single graph both descriptive and inferential statistics.

The developed statistical methods serve for the analysis of SLS algorithms on the graph colouring problem and one of its many possible generalisations, the set T-colouring problem. Several SLS algorithms have been proposed in the literature for the graph colouring problem but no "unbiased" comparison has been attempted. A similar situation holds for the set T-colouring problem. In both cases, we design new algorithms, re-implement the most prominent methods, and finally compare them in a rigorous experimental analysis. We gain indications on which are the best solution approaches for these problems and which are the best algorithms in relation to some characteristics of the instances.

As the final step, we study SLS algorithms for solving a university course timetabling problem. The design of algorithm components stems from the knowledge gained on the graph colouring problems but the assemblage and configuration of these components is carried out with a systematic methodology. The focus in this context was on the selection

of *one* single algorithm to submit to an algorithm competition on the same considered problem. The methodology is presented as an engineering process characterised by the interaction of SLS components design and empirical tests. We deem that this methodological approach is appropriate for the application of SLS methods to complex real life problems.

The main results are the following:

1. on the graph colouring problem, the simple Tabu Search with one-exchange neighbourhood remains a very competitive approach and the use of a very large scale neighbourhood is not profitable;

2. on the set T-colouring problem, the best overall algorithm is an Adaptive Iterated Greedy also based on Tabu Search with one-exchange neighbourhood which, under certain circumstances, can be further improved by a restricted exact reassignment of colours;

3. the use of an engineering methodology based on sequential testing is particularly suitable for the application of SLS methods, as it led us to devise the algorithm whose solutions for course timetabling ranked the best out of 24 feasible submissions at the International Timetabling Competition held in 2003.

Acknowledgements

This work was financially supported by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential programme of the Commission of the European Community. I also acknowledge support by the Frankfurt Center for Scientific Computing for providing access to their computer cluster where part of the experiments described in the thesis where run.

I express deep gratitude to Dr. habil. T. Stützle for his supervision. His advice, supported by the expertise in any gory detail of the Stochastic Local Search techniques guided me from the very beginning of my research activity and his invaluable comments contributed to improve considerably this work of mine. I am convinced that the impressive availability that he always showed to everybody of our research group is a rare gift for those willing to learn. Besides him, I express gratitude to Prof. Dr. W. Bibel for comments on drafts of this thesis and for leading an excellent and heterogeneous research group, though, I regret not to have taken the chance to learn more about the deductive approach of Intellectics. I am also grateful to Prof. A. Schaerf who first supervised me on timetabling problems and local search techniques during my Master's thesis and introduced me to the Metaheuristic Network.

I am deeply indebted for patience and generosity in the transmission of knowledge with all the colleagues with whom I shared the office in these three years and a half of research: Luis Paquete, in particular for bringing to my attention the permutation tests and inspiring the whole research thereafter, Tommaso Schiavinotto, for solving every practical problem that I brought to his attention, and Dr. Marco Pranzo, for increasing my interest on the problems of Operations Research.

A person who definitely showed me the pleasure and the goals of doing research is Dr. Mauro Birattari. The discussions I had with him during his stay at the Intellectics group at the beginning of my research career influenced and inspired my whole activity thereafter. His reasoning is present throughout all the pages of this work. I owe a lot also to Dr. Irina Dumitrescu with whom I carried out part of the research on the graph colouring problem. Above all, I tried to grasp from her the preciseness and rigour of the mathematical formalism.

I am also grateful to Prof. Carlos Fonseca for providing me the code to compute the empirical attainment function and to Dario Basso for his consultancy on all the issues concerned with Statistics reported in this work.

A special thank goes to the scientists who, besides Thomas, coordinated the activities of the Metaheuristic Network: Prof. Marco Dorigo, Prof. Ben Paetcher, and Prof. Luca M. Garmbardella, and to all young scientists, Dr. Monaldo Mastrolilli, Leonora Bianchi, Max Manfrin, Dr. Olivia Rossi-Doria, Krzysztof Socha, with whom I had the pleasure to work.

My thanks goes also to all the other members of the Intellectics group: Dr. Matthijs den Besten, Dr. Gunter Grieser, Dr. Hesham Khalil, Dr. Peter Grigoriev, Dr. Ulirch Scholz, Maria Tiedmann, Dr. Klaus Varrentrapp, and Dr. Sergey Yevtuschenko. A perfect group that made my stay at the Intellectics Group specially comfortable.

On a more personal basis I wish to express my sincerest gratitude to all the people

met in Darmstadt with whom I felt the rare relief of talking to a friend: Alessandro Ercoli, Mariana Forberg, Giuseppe Galluzzo, Cinzia Pagliuca, Alice Rosini, Hamid Soleymani, and Emanuela Trifan. A similar debt I owe to my flat-mates Thomas Roth, Thomas Khüner and Denise Denter and to Denis Dusso.

Still the first as the ultimate, unfailing love and support over the years was provided by my family. To them, my parents and my grandmother goes my warmest thanks.

M. C.
7th July 2005

Ah não ser eu toda a gente e toda a parte!
Ode triunfal, Fernando Pessoa

Contents

Summary vii

Acknowledgements ix

1. Introduction 1
- 1.1. Large scale optimisation . . . 1
- 1.2. Motivations and objectives . . . 3
- 1.3. Scientific publications in connection with this thesis . . . 8
- 1.4. Organisation of the thesis . . . 10

2. Stochastic Local Search Methods for Combinatorial Optimisation 11
- 2.1. Combinatorial Optimisation Problems . . . 11
- 2.2. Computational complexity and solution approaches . . . 12
- 2.3. Exact solution methods . . . 14
  - 2.3.1. Mathematical programming approach . . . 15
  - 2.3.2. Network Flow and Dynamic Programming . . . 16
  - 2.3.3. Search approach . . . 16
- 2.4. Stochastic local search methods . . . 18
  - 2.4.1. Construction heuristics . . . 20
  - 2.4.2. Iterative improvement . . . 21
  - 2.4.3. Metaheuristics . . . 23
  - 2.4.4. Hybrid methods . . . 31
  - 2.4.5. Theoretical remarks . . . 33
- 2.5. Discussion . . . 34

3. Statistical Methods for the Analysis of Stochastic Optimisers 37
- 3.1. Introduction . . . 37
- 3.2. The need for the empirical approach . . . 38
- 3.3. Application scenarios . . . 39
- 3.4. Performance measurement . . . 40
- 3.5. Statistical analysis . . . 42
- 3.6. Design and analysis of experiments . . . 44
  - 3.6.1. Experimental design . . . 44
  - 3.6.2. Statistical tests . . . 48
  - 3.6.3. All-pairwise comparisons . . . 49
  - 3.6.4. Design A: Several runs on one single instance . . . 51
  - 3.6.5. Design B: One single run on various instances . . . 55
  - 3.6.6. Design C: Several runs on various instances . . . 58
  - 3.6.7. Remarks . . . 62
- 3.7. Sequential analysis . . . 66
- 3.8. Time dependent analyses . . . 69

3.8.1. Unified representation of time and quality performance 69
3.8.2. Qualified run time distributions 74
3.9. Landscape analysis 76
3.10. Discussion 81

4. Graph Colouring **85**
4.1. Introduction 85
4.2. Formal definition of the problem and notation 86
4.3. Known theoretical results, complexity, and approximations 88
4.4. Benchmark instances and applications 90
4.5. Graph reduction 93
4.6. Exact methods 94
4.7. Construction heuristics 97
4.8. Iterative Improvement for graph colouring 102
4.8.1. Neighbourhood structure 103
4.8.2. Neighbourhood examination 106
4.9. Analysis of neighbourhood structures for local search 108
4.9.1. Analytical results 108
4.9.2. Computational analysis on small size graphs 109
4.9.3. Neighbourhood examination and heuristic rules for its speed-up . 111
4.10. Stochastic Local Search algorithms 117
4.10.1. Solving a sequence of k colouring problems 118
4.10.2. Varying the number of used colours 123
4.10.3. Extending partial colourings: a semi-exhaustive approach 124
4.10.4. Implementation Details 127
4.11. Experimental analysis on benchmark instances 128
4.12. Further analyses 137
4.12.1. On the time dependent profile 137
4.12.2. On the Tabu Search in the very large scale neighbourhood 139
4.12.3. On the Tabu Search in the one-exchange neighbourhood 139
4.13. Experimental analysis on a large set of random graphs 141
4.14. Discussion 154

5. Graph Colouring Generalisations **159**
5.1. Introduction 159
5.2. Formal definitions 160
5.2.1. Precolouring Extension 160
5.2.2. List Colouring 160
5.2.3. T-Colouring 161
5.2.4. List T-Colouring 163
5.2.5. Set T-Colouring 163
5.2.6. Related problems 164
5.2.7. Known theoretical results 165
5.2.8. Problem transformations 166
5.3. State of the art and motivations 167
5.4. Benchmark instances 168
5.5. Algorithms for the Precolouring Extension Problem 169
5.5.1. Two solution approaches 169
5.5.2. Experimental analysis 170
5.6. Algorithms for the Set T-Colouring Problem 172

5.6.1. Graph reduction . . . 172
5.6.2. Exhaustive search . . . 172
5.6.3. Lower bounds for the minimal span . . . 173
5.6.4. Construction heuristics . . . 176
5.6.5. Iterative Improvement . . . 182
5.6.6. Stochastic Local Search algorithms . . . 187
5.6.7. Experimental Analysis . . . 191
5.7. Discussion . . . 201

6. Course Timetabling **205**
6.1. Introduction . . . 205
6.2. Methods for timetabling: the state of the art . . . 207
6.3. The definition of the problem . . . 210
6.4. Timetabling and graph colouring formalism . . . 212
6.5. An engineering methodology for SLS methods . . . 216
6.5.1. The methodology . . . 217
6.5.2. Design of SLS algorithms in the UCTP-C case . . . 219
6.5.3. The racing algorithm for the experimental analysis . . . 220
6.5.4. General indications from the race . . . 221
6.6. An effective algorithm for course timetabling . . . 223
6.6.1. High level procedure . . . 224
6.6.2. Data management . . . 224
6.6.3. The assignment representation . . . 225
6.6.4. Construction heuristics . . . 226
6.6.5. Iterative Improvement . . . 227
6.6.6. Hard constraint solver . . . 230
6.6.7. Soft constraint optimiser . . . 230
6.7. Analysis of the algorithm . . . 232
6.7.1. Benchmark comparisons . . . 232
6.7.2. Contribution of algorithm components . . . 234
6.7.3. Qualified run time distributions . . . 238
6.7.4. Landscape Analysis . . . 238
6.7.5. Further analyses . . . 244
6.8. General guidelines for the application of SLS methods . . . 245
6.9. Discussion . . . 248

7. Conclusions **251**
7.1. Main themes of this thesis . . . 251
7.2. Contributions and results . . . 252
7.3. Open issues . . . 256
7.3.1. Experimental methodology . . . 257
7.3.2. A library of basic SLS components . . . 257
7.3.3. The algorithmic context . . . 258

A. A Formal Study on Iterative Improvement for Graph Colouring **261**
A.1. The conditions of local optimality . . . 261
A.2. Dominance relations between local searches . . . 265
A.3. Summary of results and discussion . . . 273

B. On the Behaviour of the Statistical Tests **275**
B.1. Implementation details . . . 275
B.2. A simulation study for general hypothesis testing . . . 277
B.2.1. Comparison of Type I Error Rates . . . 277
B.2.2. Comparison of powers . . . 279
B.3. A simulation study for all-pairwise comparisons . . . 281
B.3.1. Family-wise type I error rate . . . 281
B.3.2. Comparison of power . . . 284
B.4. Discussion . . . 286

C. Numerical Results on the Benchmark Instances for Graph Colouring Problems **287**
C.1. Graph colouring . . . 287
C.1.1. Instance statistics . . . 287
C.1.2. The parameter set used for XRLF . . . 290
C.1.3. Machine benchmark . . . 290
C.1.4. Detailed results . . . 291
C.2. Set T-colouring . . . 293
C.2.1. Instance statistics and computation times . . . 293
C.2.2. Validation of our re-implementation of tabu search . . . 293
C.2.3. Detailed results . . . 295

References **297**

"Damit Denken nicht in "Metaphysik" bzw. in leeres Gerede ausarte, ist nur notwendig, daß genügend viele Sätze des Begriffssystems mit Sinnenerlebnissen hinreichend sicher verbunden seien und daß das Begriffssystem im Hinblick auf seine Aufgabe, das sinnlich Erlebte zu ordnen und übersehbar zu machen, möglichste Einheitlichkeit und Sparsamkeit zeige."

A. Einstein. *Mein Weltbild*. 1934. Herausgegeben von Carl Seeling.

Chapter 1.

Introduction

In which we give evidence for the importance of combinatorial optimisation in the real world and point out what still has to be done for advancing the practice of applying Stochastic Local Search as effective solution method.

1.1. Large scale optimisation

Technology is offering new chances to improve human potential but at the same time it unveils new complex challenges. Logistic systems, transportation, industrial production, energy provision, communication means, administration, economic systems, marketing strategies, and investment planning are all examples where the need arises for a rational use of resources. In all cases, resources are scarce, and their use must be optimised in order to pursue a sustainable development, by controlling the impact on the environment, reducing differences in the access to technology, and finally ameliorating our conditions of life.

The optimisation of real world systems is, however, a hard task due to the dimension of these systems and to the presence of many complex constraints that must be satisfied. It becomes really feasible only with the advent of computers. Many of these real world systems can be formalised as combinatorial optimisation problems and consist in finding an assignment of discrete values to variables such that the solution is optimal with respect to some criteria. Typically, these problems are relatively easy to state but very hard to solve. In Computer Science, this concept is captured by the theory of computational complexity and the hardness of many such problems corresponds to the fact that they are $\mathcal{NP}$-hard. Because of this inherent difficulty, a large number of solution techniques for solving combinatorial optimisation problems has been proposed in different scientific fields such as Discrete Mathematics, Operations Research, and Artificial Intelligence. Often, the evaluation of these techniques through experimental investigations requires the application of Statistics for a correct analysis of experimental results. As a consequence, combinatorial optimisation is not clearly classifiable as one precise science but it can rather be characterised as being an interdisciplinary discipline.

The focus of this work is on three classes of combinatorial optimisation problems, namely *graph colouring*, some *generalisations of graph colouring*, and *timetabling*. Besides the presence of an optimisation criterion, they share the common features of having many constraints which must be rigidly satisfied by a solution. Graph colouring is a central problem in graph theory. It consists in finding an assignment of colours to vertices of

a graph in such a way that no adjacent vertices receive the same colour. In two of its many possible generalisations, defined as pre-colouring extension and set T-colouring problems, a part of the graph is already coloured or specific separations must be satisfied between colours assigned to adjacent vertices. These problems are an abstraction of many real world problems, where less important details are removed and the essence of the problem type is captured. Such abstract problems allow techniques and analyses to be more general and, thus, be more adaptable as the underlying application details changes.

Practical applications that can be modelled as graphs colouring problems are all cases where the assignment of variables is "constrained" in a specific way by the assignments of some other related variables. Examples are scheduling problems, which involve restrictions in processing two jobs at the same time; register allocation during the execution of a computer program, where constraints capture the fact that some variables cannot be assigned to the same register; or the management of air traffic flow, where the goal is optimising the capacity of the airspace while satisfying vertical separations between intersecting flight routes.

Besides these, an important field of application for graph colouring is the assignment of channels in telecommunications. Graph colouring in its simplest formulation can be used to solve subproblems of a more complex network design. Examples have been presented in the assignment of light wavelengths in transparent optical networks. More commonly, however, the assignment of frequencies involves some other constraints that complicate the problem. These constraints can be modelled by the generalisations of graph colouring problems into set colouring problems, T-colouring problems and similar ones. This allows to formalise, and solve, for example, the assignment of several frequencies to each cell of a mobile network by satisfying constraints of minimal interference between adjacent cells.

Finally, graph colouring is at the core of all typologies of timetabling. In all such applications, a recurrent constraint is that events that share resources cannot be scheduled in the same time period. As an example, in university course timetabling, courses that are taught by the same professor or are attended by the same students cannot be scheduled in the same timeslot. Timetabling problems arise in many contexts besides university. Examples are railways timetabling, employee shifts timetabling, aircraft scheduling, and sport timetabling. They comprise peculiar constraints and a universal method for solving all of them is hardly feasible.

The techniques available for solving these combinatorial optimisation problems fall into two main classes: *exact* and *approximate*. Whereas the former can prove the optimality of the solutions found when given enough time, the latter cannot but typically provides better solutions for short computation times. Typically, exact methods are limited to very small problems and are not practicable for large scale real world systems. The need then arises to circumvent these limitations by accepting solutions of not provable quality. Approximate *heuristic methods* are the best answer to this need. They are particularly appealing because of their flexibility, easiness of implementation, and capability of attaining satisfactory performance on many kinds of optimisation problems even of large dimensions and with several constraints. Important journal publications in the field of Operations Research (European Journal of Operational Research, Journal of Scheduling, Journal of Heuristics, Management Science, INFORMS Journal of Computing) clearly indicate that these methods are appealing and successful for practical problems. The focus of this thesis is on the application of these methods, specifically stochastic local search methods, for the solution of the three problems mentioned above.

> *Stochastic local search* (SLS) methods are heuristic methods for combinatorial optimisation problems developed in Operations Research and Artificial Intelligence. The process of *searching* proceeds by trial and error. It consists in iteratively examining sequences of close solutions of known value and taking the best of these sequences. This process is an informed search process as it exploits problem-specific knowledge. Nevertheless, the kind of information used is *local*, in the sense that the alternatives which are examined are not recorded at each point during the search and the search is not systematic. Finally, the search process is *stochastic* because it is based on probabilistic decisions. Random decision are introduced whenever it is unknown which deterministic rules would allow to reach good solutions for all the problem instances of interest.
>
> SLS methods are obtained by assembling different components. We will distinguish three main components: construction heuristics, iterative improvement, and metaheuristics.

SLS methods are currently recognised as state-of-the-art methods for many combinatorial problems, as they are efficient in yielding high-quality solutions in relatively short time and they are easily adaptable to different problems. Their use in the field of Operations Research is now well established. Nevertheless, in the last 20 years several variants of these techniques appeared in scientific publications spreading confusion about their use. As a consequence, in recent years, we assisted an arising consciousness about the need of classifying these techniques under common frameworks and defining a theory of practice for their application. While the first task appears now being accomplished with important publications such as the book of Hoos and Stützle (2004) or the framework by Di Gaspero and Schaerf (2003a), the second remains still unclear.

Our point of view in this thesis is that design, development, and evaluation, are part of a unique algorithm engineering process for the application of SLS methods. Chiefly important in this process is the use of experiments and appropriate tools for the analysis to lead to correct decisions in the design and implementation of high-performing algorithms. As in science, the final goal is to "order and make predictable the real world through a system of concepts with highest unity and parsimony"[1]; thus, in algorithm engineering, the goal is developing optimisation algorithms that are effective and efficient, and use only indispensable components. The definition and adoption of a systematic methodological procedure is necessary to achieve this goal.

1.2. Motivations and objectives

According to the philosopher Popper (1963), a young scientist who hopes to make discoveries in science should "try to learn what people are discussing nowadays in science; find out where difficulties arise, and take an interest in disagreements; these are the questions he should take up." In other terms, the problem situation of the day should be studied and the line of inquiry, which is based on the background of the field, continued. What has already been done serves as a system to which we refer. We use it by checking it

[1] The passage, quoted from Einstein "Mein Weltbild", 1934, is a free English translation of the author. The original version is reported at page xv of this thesis.

over, and by criticising it. In this way, science makes progress and in this way we hope to advance the application of SLS methods.

SLS algorithms have behind a considerable body of literature in Operations Research, Artificial Intelligence and Mathematics (Papadimitriou and Steiglitz, 1982; Aarts and Lenstra, 1997; Glover and Kochenberger, 2002a; Russell and Norvig, 2003; Hoos and Stützle, 2004). Many different methods and small variants thereof have been proposed. In more recent years the space for seminal ideas has drastically decreased and the interest has shifted towards the organisation, refinement and selection of existing methods. As scientists in this field, our interest is in understanding, articulating, and extending the paradigm underlying these algorithms that already exist and in deriving a theory of practice for their implementation. Despite some attempts through landscape analyses and theoretical insights on the convergence of the algorithms, many issues on these algorithms remain open. While an answer to these issues through theoretical proofs remains very hard, new consciousness arose that the experimental method can provide satisfactory answers. Our interest is in determining whether these methods, as they have been applied so far, yield the best ever possible performance or if margins still exist for improvements. Most important, we would like to clearly map SLS methods to optimisation problems, that is, having a problem and specifications on its instances, we would like to find in the literature (or more likely in the Internet) a clear answer about which algorithm is worth to be re-implemented as it has been shown to perform the best among many others. A similar map could be worth also for the components underlying SLS methods, such as, construction heuristics, neighbourhood structures, local search procedures, number of solutions maintained, the interaction between these solutions, etc. Whenever a pratictioner would face the same problem, or a similar one, he could consult these results and adopt the correct method.

The Metaheuristic Network, a Research Training Network within the Improving Human Potential Programme of the European Commission, which involved five research groups and two companies and within whose framework the author of this thesis carried out his research, had as initial goals answering in part these issues. However, it became clear with the time that trying to extrapolate basic principles for each SLS component, in particular for the high level general purpose metaheuristics, independently from the other components of an SLS method was leading to conclusions of rather weak practical relevance. It emerged instead that matching the problem details and using the synergetic properties of SLS components was by a large order of magnitude more important than the impact of high levels metaheuristic components. New awareness arose that the success of an application of SLS methods is the outcome of an engineering process with experimental analysis in its central place, in which all components and their combinations must receive attention.

Unfortunately, the fact that the assessment of stochastic optimisation algorithms is a possible field for applied statistics, with significant problematic issues to be solved, has been so far partially neglected. There is definitely a need for understanding which tools of statistics are appropriate in the context of SLS analysis and which could alleviate the task to SLS practitioners. In fact, not one single tool is needed but rather a set of tools for accomplishing different tasks such as comparisons, predictions, scalability studies, combined analysis of multiple criteria, search landscape analyses, and classifications of algorithms. This necessity is caught by some important conferences, like the International Workshop on Efficient and Experimental Algorithms, books, like those of Cohen (1995) and Fleischer et al. (2002), journals like the Journal of Experimental Algorithms, and many articles (Hooker, 1996, Barr et al., 1995, etc.).

The main contributions of this thesis in the line of research outlined above can be summarised as follows.

Statistical methodologies for the comparison of algorithms

The practice of using statistical tests for supporting the comparisons between optimisers is not yet widely established. Several publications state the importance of using such methods. Nevertheless, a clear definition of the experimental scenarios and the methods appropriate for the analysis is rarely provided (few important exceptions are Rardin and Uzsoy (2001); Johnson and McGeoch (2002); Fonseca and Fleming (1996); Birattari et al. (2002)). Researchers in the field of optimisation are thus left with the duty of searching through the text-books of statistics for those correct methods.

We intent to rise the concern here that some of the statistical assumptions underlying the most common methods of analysis are inadequate for the analysis of experimental designs for comparing optimisation algorithms. In particular, we show that parametric assumptions are often not appropriate and we give details on alternative non-parametric methods. We implement permutation tests and compare them with rank-based tests in different experimental designs for practical cases of the comparison of optimisers and we show that they can lead to different inferences. We also extend a known visualisation method based on confidence intervals for reporting the results of the analysis to recent non-parametric methods. This representation allows for an immediate and easy perception of significant results.

The methodology thus defined will be used for the evaluation of SLS algorithms for the three problems of our concern.

Comparison of algorithms for graph colouring problems

Some problems are more important than others because of their generality and the interest they raised. These problems constitute a core class of problems which often arise as sub-problems in real-world applications. They do not exhibit too many technicalities and allow to perform case studies that, besides the outcome for the specific problem, help to develop a methodology useful also in more complex situations.

We study the classical graph colouring problem, the set T-colouring problem and dedicate a marginal attention to pre-colouring extensions. Our interest was raised by the Graph Colouring Symposium where many issues remained open and attracted the curiosity of the author. *Our aim is the empirical characterisation of the behaviour of effective approximate algorithms on problem instances where exact methods become infeasible.* Similar analyses have been done on other important problems like, for example, the TSP (Johnson and McGeoch, 2002; Johnson et al., 2002a) or, for search algorithms, on the bipartite matching problem (Setubal, 1996), and constitute very helpful references for all those who afterwards have to solve the same, or similar, problems in practice.

These issues are addressed in Chapters 4 and 5 and, the main contributions can, in some more details, be summarised as follows.

Algorithms for the graph colouring problem. We re-implement and test some of the SLS algorithms for the GCP for which particularly positive results were reported. For the same level of performance, preference is given to approaches that are better known and that require less effort in the implementation. Lower relevance is also

given to algorithms that require the tuning of many parameters, since this may be a cause for scarce robustness.

In addition to the known algorithms, we design and test new algorithms for the two problems and adapt metaheuristics like Iterated Local Search, Guided Local Search, the Min-Conflicts heuristic, and Adaptive Greedy Search. The choice of these algorithms is due to the fact that they are currently state-of-the-art on problems that are very similar to graph colouring such as constraint satisfaction problems and satisfiability problem in propositional logic.

A concern of main interest is understanding whether the whole potential of stochastic local search methods has been exploited for the graph colouring problem. To this end, we devote particular attention to the examination of neighbourhoods structures which are at the core of every SLS algorithm. Complex neighbourhood structures boosted the performance of local search algorithms on problems like the travelling salesman problem or the set partitioning problem. We try to do the same on the graph colouring problem comparing the use of small against large neighbourhoods that can be searched effectively.

Experimental Analysis. The challenge launched in the context of the Graph Colouring Symposium was partly inconclusive. It was not possible to state clearly which algorithm was better than others on which class of instances. Above all, the comparison of new against old algorithms known from the literature was very hard due to the use of different experimental settings and the way results are presented. Our criticism to these comparisons is that they are often (wrongly) based on best results and on the frequency with which they are attained. Results are often discussed on the basis of few instances and doubts arise that such results are the consequence of an over-tuning or simply a matter of chance and, in both cases, not relevant for practical inference. Moreover, when comparing algorithms described in different publications, it is very unlikely that all algorithms are considered under the same experimental conditions.

In this thesis, we aim at an "unbiased" comparison, that is, a comparison in which all algorithms are allowed to use the same computational resources and work under the same conditions, that is, use the same data structures and equal effort for tuning. The methods of analysis defined in Chapter 3 are applied for the inference of statistically correct conclusions and for the graphical representation of the results.

Most experimental analyses on graph colouring report results on a set of about 120 instances which are the same since over 10 years. We show that many of them are actually easy and not challenging enough for comparing high-performing approximate algorithms. Moreover, we maintain the analysis of algorithms separated for classes of problem instances which are recognisable *a priori*, thus deepening the understanding of the problem. Besides basing our analysis only on challenging instances, we also extend our study to a new set of random instances. The large number of new instances generated allows for the reduction of the variance of the estimated performance. Moreover, randomly generated graphs also permit to control some inherent features of the graphs thus allowing study the specialisation of the algorithms on instance classes.

In addition to the comparison of algorithms, we examine their behaviour over time trying to understand when it is reasonable to stop them. For some algorithms we try to understand which are the reasons that make them perform well or poorly by

analytical, detailed profiling, and the recognition of "pathological" cases.

Graph colouring generalisations. A disappointment at the Graph Colouring Symposium was the scarce interest given to the graph colouring generalisation. The possible generalisations are manifold. We focus on one in particular: the set T-colouring problem. This problem has a relevant practical application which is frequency assignment. We unify these two areas which so far have been maintained separated, transferring and testing ideas published in the two fields.

Comparisons of SLS algorithms for the set T-colouring or for the frequency assignment problem have been even more problematic than in classical graph colouring. The contribution of a systematic analysis similar to the one developed for the classical graph colouring appears therefore very timely in this context.

Similar to classical graph colouring, we thoroughly study SLS components, such as construction heuristics, and neighbourhood structures. Above all, we introduce a new neighbourhood in the local search which solves sub-problems in an exact manner. Finally, we compare effective SLS algorithms and study their computational effort.

The study on the set T-colouring problem serves also to test for basic guidelines in the application of SLS algorithms to problems with many constraints. In particular, we intend to make clear, through empirical tests, whether it is more appropriate to relax constraints or to use them for restricting the search space.

A methodology for the application of SLS algorithms to real-world problems.

When working on real life problems like timetabling the situation is slightly different from the one in the previous point. Typically, real-world problems have highly specific constraints linked to the context where they arise and the chance of finding in the literature a case with the very same characteristics is minimal. The focus then is less on specific algorithms but rather on the methodological procedure that lead to the development of a highly effective SLS optimiser for the specific application.

We present in this case the successful application of a recently introduced racing methodology (Birattari et al., 2002) based on sequential testing. The methodological framework developed at the first point entitles us to adapt a sequential testing procedure to the different scenarios that may appear in a real application context. Sequential testing implements a race between algorithms for determining the best algorithms by running only a small amount of experiments. The reduction of the computational effort in the testing phase is profitable for two tasks: (i) gathering quickly essential pieces of information on the best approaches to solve the problem and redirecting the design of improved algorithms only towards these promising approaches; (ii) inclusion of many configurations thus allowing for a better tuning. We use the racing methodology in an interactive procedure for devising the algorithm for the International Timetabling Competition.

We describe our experience and argue that the application of SLS methods to solve new complex real life problems necessarily progresses by guesses and by tentative solutions. A precise, fast, and objective testing methodology to compare different approaches is therefore particularly suitable. In contrast, at least for the case studied here, other methods of analysis such as analytical studies or landscape analysis failed to provide any relevant information for the practice.

The application of a systematic methodology allows us to apply to the timetabling problem several different construction heuristics, local searches, and best known metaheuristics, and to severely test them. Besides being profitable for the development of a good algorithm, this situation offers us the possibility to infer guidelines for future applications of SLS methods.

1.3. Scientific publications in connection with this thesis

The work described in this thesis is also object of articles or reports that the author, together with other co-authors, has published, submitted for publication, or are currently under preparation for submission to journals or conferences.

An extract of the definition of experimental scenarios and statistical methods for the comparison of the performance of algorithms presented in Chapter 3 is given in

> M. Chiarandini, D. Basso, and T. Stützle. (2005). "Statistical methods for the comparison of stochastic optimizers". In *Proceedings of the Sixth Metaheuristics International Conference, MIC2005.* Vienna, Austria.

An application of these methods for the analysis of algorithm components and for the comparison of hybrid algorithms in the context of the Vehicle Routing Problem with Stochastic Demand is described in

> L. Bianchi, M. Birattari, M. Chiarandini, M. Manfrin, M. Mastrolilli, L. Paquete, O. Rossi-Doria, and T. Schiavinotto. (2005). "Hybrid Metaheuristics for the vehicle routing problem with stochastic demand". *Journal of Mathematical Modelling and Algorithms*. Accepted for publication.

The interest for graph colouring arose as a consequence of an application of Iterated Local Search for solving the problem presented at the Graph Colouring Symposium in 2002:

> M. Chiarandini and T. Stützle. (2002). "An application of Iterated Local Search to Graph Coloring". In *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, edited by D. S. Johnson, A. Mehrotra, and M. Trick, pages 112–125, Ithaca, New York, USA.

The study on very large neighbourhoods for Iterative Improvement on graph colouring, which occupies a large part of Chapter 4 and Appendix A, is based on:

> M. Chiarandini, I. Dumitrescu, and T. Stützle. (2003). "Local Search for the Graph Colouring Problem. A Computational Study". Technical Report AIDA–03–01, Fachgebiet Intellektik, Fachbereich Informatik, Technische Universität Darmstadt.

The study on effective stochastic local search methods for graph colouring will be the object of a book chapter under preparation:

> M. Chiarandini, T. Stützle, and I. Dumitrescu. (2005). "Stochastic Local Search Algorithms for the Graph Colouring Problem". In *Approximation Algorithms and Metaheuristics*, edited by T .F. Gonzalez. Computer & Information Science Series, Chapman & Hall/CRC. To appear.

while the analytical results on the local search discussed in Appendix A is planned to become part of a separate report:

> M. Chiarandini, L. Paquete, I. Dumitrescu, and T. Stützle. (2005). "A formal study on local search for the graph colouring problem". To be submitted to *Discrete Applied Mathematics*.

The whole Chapter 5 on the generalisation of graph colouring is new. Its content will be collected in an article:

> M. Chiarandini and T. Stützle. (2005). "Stochastic local search for graph colouring generalisations".

The work on the timetabling described in Chapter 6 is based on a tight collaboration with other researchers and linked to the activities of the Metaheuristic Network. A preliminary analysis (not reported in this thesis) of run time distributions for algorithms that find feasible solutions, on the basis of the general concepts described in Chapter 3, is reported in

> M. Chiarandini and T. Stützle. (2002). "Experimental evaluation of course timetabling algorithms". Technical Report AIDA–02–05, Fachgebiet Intellektik, Fachbereich Informatik, Technische Universität Darmstadt.

The attempt to compare basic stochastic local search algorithms that use different metaheuristics is described in a joint publication of the members of the Metaheuristics Network:

> O. Rossi-Doria, M. Samples, M. Birattari, M. Chiarandini, M. Dorigo, L. Gambardella, J. Knowles, M. Manfrin, M. Mastrolilli, B. Paechter, L. Paquete, and T. Stützle. (2003). "A comparison of the performance of different metaheuristics on the timetabling problem". In *Practice and Theory of Automated Timetabling IV: Selected Revised Papers*, edited by E. Burke and P. Causmaecker, volume 2740 of *Lecture Notes in Computer Science*. Springer Verlag, 2003. Extended abstract available on the Proceedings of PATAT 2002, pages 115-119.

The algorithm that won the International Timetabling Competition is also described in the two following works:

> M. Chiarandini, K. Socha, M. Birattari, and O. Rossi-Doria. (2003) "International timetabling competition. A hybrid approach". Technical Report AIDA–03–04, Fachgebiet Intellektik, Fachbereich Informatik, Technische Universität Darmstadt.
>
> M. Chiarandini, M. Birattari, K. Socha, and O. Rossi-Doria. (2005) "An effective hybrid approach for the university course timetabling problem". *Journal of Scheduling*. Accepted for publication.

Finally, the author participated also to the ROADEF 2005 as a member of a team of young researchers. Object of the competition was a car sequencing problem. The development methodology and the final algorithm is very similar to the one described in Chapter 6 for the timetabling case. This experience is reported in

> M. Risler, M. Chiarandini, L. Paquete, T. Schiavinotto and T. Stützle. (2004). "An algorithm for the car sequencing problem of the ROADEF 2005 Challenge". Technical Report AIDA–04–06, Fachgebiet Intellektik, Fachbereich Informatik, Technische Universität Darmstadt.

1.4. Organisation of the thesis

The thesis continues in Chapter 2 with an introduction to combinatorial optimisation and solution methods. This chapter defines the characteristics of an optimisation problem and introduces the terminology used throughout this thesis. It provides a comprehensive picture of the available algorithmic approaches and places stochastic local search methods in this context. Here, the components of stochastic local search methods, construction heuristics, local search, and metaheuristics are described and algorithmic sketches are provided.

The analytical analysis of stochastic local search algorithms is very hard and the empirical way has been recognised as more suitable for providing results of practical relevance. Chapter 3 is concerned with the analysis of experiments and the methodology for the comparison of stochastic optimisers. It introduces the statistical concepts needed and indicates how to use them for the analysis of algorithms.

The graph colouring problem is the object of Chapter 4. The comparison of several approximate algorithms is carried out on challenging, large benchmark instances and newly generated random instances. The presentation proceeds from construction heuristics, through iterative improvement algorithms, to the best performing Stochastic Local Search algorithms. The best known algorithms in the literature are described and reimplemented. In addition, new algorithms like Novelty, Min-Conflict heuristic, Guided Local Search, are here introduced. In particular, extensive analysis is devoted to a new very large scale neighbourhood for local search.

In Chapter 5 we extend the results of the previous chapter to graph colouring generalisations. The main focus is on the T-colouring and set T-colouring problems, although results for pre-colouring extensions are also provided. The chapter follows the same structure of the previous one by describing and analysing construction heuristics, local search, and stochastic local search methods. Precise questions are posed on the best way to approach these problems characterised by the presence of many constraints of different nature. The questions are then answered by an empirical analysis and the best performing SLS algorithms are determined.

Chapter 6 is concerned with the application of the methods and methodologies developed in the previous chapters to a real-world context. The timetabling problem is strictly related to the graph colouring problem but its complexity, due to the presence of several side constraints and objective criteria necessarily requires a slightly different approach than the one followed for the two previous problems. We recognise that the process for the development of a highly performing algorithm is an engineering process. Here we describe in detail the application of a racing methodology as a tool for helping in this engineering process. The fact that our final algorithm submitted to the International Timetabling Competition ranked best, demonstrates the usefulness of our procedure. The algorithm is described in detail and its essential components are analysed. Finally, guidelines for the application of SLS methods arising from this experience are presented.

Chapter 7 closes the thesis with a summary of the main contributions and directions for further research.

The appendices report about additional results. Appendix A contains the formal analysis of the neighbourhoods structures for graph colouring, Appendix B contains a simulation analysis for the comparison of the statistical tests presented in Chapter 3, and Appendix C reports numerical results for cross-checking with other literature on the graph colouring problem (Chapter 4) and the set T-colouring problem (Chapter 5).

Chapter 2.

Stochastic Local Search Methods for Combinatorial Optimisation

In which we introduce combinatorial optimisation problems and present the techniques available to solve them. The focus is on stochastic local search methods.

2.1. Combinatorial Optimisation Problems

A problem is a general question to be answered. It can be completely described by giving an *input*, *i.e.*, a general description of the problem parameters, and a *question*, *i.e.*, the statement of what properties the answer, or *solution*, must satisfy. The specification of particular values for all problem parameters is an *instance* of the problem. A *search problem* consists of a set of instances $\mathcal{I}$ with a set of possible answers $\mathcal{S}(I)$ and a finite set of *solutions* $S(I) \subseteq \mathcal{S}(I)$ for each instance $I \in \mathcal{I}$. An algorithm is said to solve a search problem if it correctly determines that $S(I)$ is empty, or otherwise, it returns some solution $s \in S$ which constitutes a correct answer to the problem. An *optimisation problem* is a search problem in which one or more *preference criteria* are defined to rate solutions in $S(I)$. It can be formalised as either a *minimisation problem* or a *maximisation problem* and consists of:

- a set of instances $\mathcal{I}$;
- a finite set of *feasible candidate solutions* S for each instance $I \in \mathcal{I}$, and
- an *objective function* g that assigns to each candidate solution $s \in S(I)$, $I \in \mathcal{I}$, a positive, real number $g(I, s)$ called *solution value* or *objective function value* of s.

A combinatorial optimisation problem is an optimisation problem in which candidate solutions are defined by variables that may only assume integer values. Throughout this thesis, we assume to deal with minimisation problems unless explicitly specified. Furthermore, we also use the terms *quality* of a solution to denote its assessment with respect to the minimisation criterion, and *cost function* and *solution cost* as synonymous for objective function and solution value, respectively.

In optimisation problems, it is common practice to distinguish general characteristics of the answer, which define the set of *candidate solutions* $\mathcal{S}$, from constraints, which restrict the set of candidate solutions to the set of *feasible candidate solutions* S. The difference between solution characteristics and constraints may be subtle and may depend on the

way the problem is modelled. We illustrate this difference with the example provided by the constraint satisfaction problem, a problem at the core of those studied in this thesis.

The *constraint satisfaction problem* (CSP) consists of a set of variables $X_1, X_2, \ldots, X_n$ with a non empty domain $D_1, D_2, \ldots, D_n$ of possible values. A set of constraints $C_1, C_2, \ldots, C_m$ involves some subset of the variables and specifies the allowable combinations of values for that subset. The search problem asks to find an assignment $\{X_1 = v_1, X_2 = v_2, \ldots, X_n = v_n\}$ of values to variables from their respective domains such that all constraints are satisfied. The optimisation problem asks to find, among the assignments which satisfy all constraints, the ones that are the most desirable according to some preference criterion.

In the CSP, we may conceive at least three different ways to define the set of candidate solutions. It may consist of all possible complete assignments of values to the variables from their respective domains, while the set of feasible candidate solutions being restricted to the assignments that satisfy all the constraints C_i; or it may comprise also partial assignments in which the values for some variables is still undefined; or it may be even larger and comprehend also assignments of values to the variables that do not come from their respective domains, using then the sets of domains at same level as the constraints for defining the set of feasible candidate solutions. In the rest of the thesis we refer to a *candidate solution* as *solution*, for short.[1] We will, however, define for each problem the solution representation adopted.

The preference criteria are in fact different kind of constraints, which limit the set of feasible solutions to the set of preferred solutions. In the language of Operations Research, the constraints that specify the set of feasible solutions are called *hard constraints* while the constraints that specify the function to minimise are called *soft constraints*. In this thesis, we adhere to this terminology and denote the first with H and the second with Σ.

2.2. Computational complexity and solution approaches

In general, COPs have the characteristic of being very easy to state but very hard to solve. The concept of hardness of a problem is formalised by the computational complexity theory and in particular the theory of $\mathcal{NP}$-hardness. Accordingly, search problems are reduced to decision problems by shifting the interest from finding a solution to determining whether a solution with given properties exists. Decision problems are then classified in relation to existing algorithms for solving them. An algorithm can be characterised by a *time complexity function*, which indicates, for each possible input length, the largest amount of time (or steps) needed by the algorithm to solve an instance of that size. We say that a function $f(n)$ is $\mathcal{O}(h(n))$ whenever there exist integers c and N such that $|f(n)| \leq c \cdot |h(n)|$ for all values of $n \geq N$.[2] A *polynomial time algorithm* is then an al-

[1] The term is improper because a candidate solution may also be infeasible and hence not be a "solution" of the problem. Nevertheless, this terminology is commonly adopted in the field of stochastic local search methods for optimisation. To denote then the real "solution" of the problem the identifier "optimal solution" is adopted.

[2] In this thesis we will use the notation $\mathcal{O}(\cdot)$ for the asymptotic worst case behaviour of an algorithm. The asymptotic mode removes confusing behaviour due to start-up costs. In a similar vein, $f(n) = \Omega(h(n))$ whenever there exist integers c and N such that $|f(n)| \geq c \cdot |h(n)|$ for all $n \geq N$ will be used as regard to an asymptotic best case behaviour.

gorithm whose time complexity function is $\mathcal{O}(p(n))$ for some polynomial p. We call any other algorithm, whose time complexity function cannot be bounded by a polynomial, *exponential time algorithm*. Decision problems for which polynomial algorithms exist are called *tractable* and are grouped in the class $\mathcal{P}$; those for which no polynomial time algorithm is known, are referred to as *intractable* and are grouped in the class $\mathcal{NP}$. In simple words, supposed intractable problems are so difficult that, in the worst case, an exponential amount of time is needed to discover a solution. It is straightforward to show that $\mathcal{P} \subseteq \mathcal{NP}$.[3] Even though it is widely believed that $\mathcal{P} \neq \mathcal{NP}$, at the time of writing, no proof for this conjecture has been found.[4] All problems in $\mathcal{NP}$ can be reduced in polynomial time to a group of problems which constitute the class of $\mathcal{NP}$-complete problems. Problems in this class are the hardest problems in $\mathcal{NP}$ because showing that one of these problems is tractable would mean showing that all problems in $\mathcal{NP}$ are tractable. Finally, the notion of $\mathcal{NP}$-hard problems is used to classify all those problems which are not in $\mathcal{NP}$ but problems in $\mathcal{NP}$ can be transformed to them in polynomial time. This more general class of problems comprises, for example, the search and optimisation variants of the $\mathcal{NP}$-complete problems.

Optimisation problems which are $\mathcal{NP}$-hard may still have important special cases that are solvable in polynomial time. However, if we are faced with a case that is not among them, there are two general approaches for solving $\mathcal{NP}$-hard problems.[5]

A first possibility, even recognising that an algorithm will inevitably require exponential time, is to try to obtain as much improvement over exhaustive search as possible. The best known approaches to reduce the search effort are implicit enumeration techniques based on branch and bound, backtracking, and dynamic programming. Other approaches, that allow to organise effectively the search, are mathematical linear programming with the use of cutting planes or Lagrangian techniques. One of these approaches may be perfectly satisfactory if the actual input size of the instance is small.

A second possibility is to no longer focus on finding an optimal solution and proving its optimality but to try, instead, to find a "good" solution within an acceptable amount of time. Algorithms that try to do this are loosely termed *heuristic algorithms* since they are mainly based on common-sense rules. Heuristic algorithms tend to be rather problem specific and require to be fine-tuned in order to achieve high performance. For these reasons, formal analyses are rare and comparisons are done on an empirical basis.

There are, however, heuristic algorithms for which formal results are known. This is the case of *approximation algorithms*: they provide a solution to a problem in polynomial time with a certain performance guarantee. Let A be an approximation algorithm that

[3] The definition of classes $\mathcal{P}$ and $\mathcal{NP}$ is much more formal in computational complexity theory. Problems are represented as "languages" and typically the one-tape Turing Machine is used as a model for computation. The class $\mathcal{P}$ is then defined as all languages recognisable deterministically in polynomial time, while the class $\mathcal{NP}$ is defined to consist of all languages recognisable "non-deterministically" in polynomial time. Non-deterministic algorithms can be imagined as being composed of two stages, a guessing stage which provides some "structure" and a checking stage which computes deterministically whether the structure answers the question posed. The non-deterministic algorithm is said to operate in polynomial time, if the checking stage is accomplished in a polynomially bounded amount of time.

[4] There is a prize of 1.000.000$ awarded by the Clay Mathematics Institute (CMI) for a correct solution of the $\mathcal{P}$ versus $\mathcal{NP}$ problem, while a collection of published and unpublished papers on this issue is available at `http://www.win.tue.nl/~gwoegi/P-versus-NP.htm` (November 2004).

[5] Throughout this thesis we distinguish between *approaches*, *methods* (or techniques), and *algorithms*; in increasing order of specificity, an approach is a general framework for the development of a solution algorithm (*e.g.*, exact or approximate solution, mathematical modelling, complete partial candidate solution representation), a technique is an algorithmic procedure or a set of algorithmic procedures for the solution of a problem (*e.g.*, integer linear programming, construction heuristic, local search), an algorithm is the final instantiation of a technique for solving a specific problem.

returns a candidate solution s when applied to instance I and let $A(I)$ be the solution value $f(I, s)$. If we denote with OPT(I) the optimal solution value, we can define the ratio $R_A(I)$ as $R_A(I) = A(I)/\text{OPT}(I)$. The *absolute performance ratio* R_A for an approximation algorithm A is then the largest ratio $R_A(I)$ over all possible instances of the problem. Similarly, the *asymptotic performance ratio* is the largest ratio $R_A(I)$ over all instances of the problem satisfying OPT$(I) \geq N$ for some $N \in Z^+$.[6] There are problems for which polynomial time algorithms are known which provide a more or less good performance ratio. There are, however, also problems where formal proofs exist that, unless $\mathcal{P} = \mathcal{NP}$, no polynomial time approximation algorithm can guarantee a performance ratio below a certain fixed threshold. An example of such a problem is the graph colouring problem. A polynomial time graph colouring algorithm A with $R_A \leq 4/3$ would have to colour any 3-colourable graph with no more than 3 colours and hence would solve the graph 3-colourability problem in polynomial time. But the graph 3-colourability problem is an $\mathcal{NP}$-complete problem, thus if $\mathcal{P} \neq \mathcal{NP}$, no such approximation algorithm can exist. We refer to Section 4.3 on page 88 for more results on the approximability of the graph colouring problem.

As opposed to approximation algorithms, we denote simply as *approximate algorithms* those heuristics that return near-optimal solutions in a relatively short time but for which no formal prove on their performance guarantee has been found so far. The performance of approximate algorithms is typically described by empirical studies.

The literature on combinatorial optimisation is vast. A reference book in combinatorial optimisation, which encompasses methods, problems, and computational complexity, remains the book of Papadimitriou and Steiglitz (1982), although published more than 20 years ago. Foundations of computational complexity in combinatorial optimisation are given by Garey and Johnson (1979) and Papadimitriou (1994). For a thorough coverage of approximability theory we refer to the books of Vazirani (2001) and of Ausiello et al. (1999),[7] while Hochbaum (1996) collects specialised papers.

2.3. Exact solution methods

Exact methods guarantee to find an optimal solution and to prove its optimality for every finite size instance of a COP within an instance-dependent run time. In the worst case, their time complexity function is exponential. Within exact methods one can distinguish two classes of approaches: general purpose and problem specific ones. In the first group there are the methods based on mathematical programming in which all problems are modelled using, for example, a set of linear equations and the resulting linear system is solved with techniques from linear programming, a branch of Mathematics. In the second group there are the methods based on the idea of intelligent enumeration of so-

[6] A further refinement in approximability concepts is the definition of *polynomial time approximation schemes* in which an approximation algorithm defines a range of approximation algorithms, one for each fixed value of an accuracy requirement $\epsilon > 0$. These algorithms take as input an instance I and an accuracy requirement ϵ and return a solution such that $R_A(I) \leq 1 + \epsilon$. Their time complexity function is bounded by a polynomial in the instance size but, typically, exponential in $1/\epsilon$ (see Schuurman and Woeginger, 2001 for a publically available tutorial). In practice, the time complexity of such algorithms to obtain close-to-optimal solutions is often bounded by a polynomial with rather high degree (especially as $\epsilon \to 0$ we have that $1/\epsilon \to \infty$).

[7] The book has an updated catalogue available on the Internet of approximability results for $\mathcal{NP}$ optimisation problems (Crescenzi and Viggo, 2004)

lutions using problem specific knowledge. The most effective methods within this class are network flow and dynamic programming but their application is effective only on a restricted group of well known problems. Search methods, mainly from the field of Artificial Intelligence, are, instead, more flexible.

2.3.1. Mathematical programming approach

In mathematical programming, a common approach is to formalise an optimisation problem as a system of linear equations in real variables which is usually written in the form

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} c_j x_j \\ \text{such that} \quad & \sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad i = 1, \ldots, m \text{ and } b > 0 \\ & x_j \geq 0 \qquad\qquad\quad j = 1, \ldots, n \end{aligned} \tag{2.1}$$

Linear programming techniques for solving this system are fast and efficient. In addition, there exists a wide variety of available software and good modelling languages. Solvers are based on the simplex method combined with the primal-dual algorithm (which provides upper bounds on the optimal value), or, less frequently, on the interior point method or the ellipsoid method (this latter having a polynomial worst case running time). We refer the reader to Carter and Price (2000) for a thorough treatment of these techniques or to the web-site of the Network Enabled Optimisation System (NEOS)[8] for an interactive demonstration of the simplex method as well as for a comprehensive guide to optimisation models, algorithms and software.

Integer Linear Programming (ILP) corresponds to a linear program where the x_j variables of System 2.1 are integer. All problems treated in this thesis can be formalised as integer linear systems. Also ILP presents convenient modelling possibilities and available software, but its performance varies: some large instances can be solved easily, while some other small instances cannot be solved within even a large amount of time. In ILP, constraints are first pre-processed to identify redundancy, infeasibility and to tighten the bounds. Then, the ILP system is transformed into its linear programming relaxation by allowing integer variables to assume real values. Once the problem is in linear programming form, the basic solution method uses *branch and bound* strategies to retrieve the best integer solution. The original problem is split into sub-problems subdividing the solution space according to some branching rules and giving rise to a binary tree. Sub-problems are solved following some order (depth-first or breadth-first) and nodes are fathomed (pruned) by considerations on the bounds provided by the solutions of the sub-problems. The procedure ends when a solution with all integer values has been found and no other node remains to be branched. This algorithm is, then, strengthened by combining it with other techniques. *Lagrangian relaxation* simplify the system by inserting the most "complicated" constraints in the objective function penalising them by some multipliers. *Cutting plane* algorithms reduce the space of continuous solutions without loosing integer solutions. To this end, valid constraints that do not exclude integer feasible points are added to the linear inequalities of the problem. Despite the fact that inequalities are problem

[8] Argonne National Laboratory and Northwestern University. "Optimization Technology Center". November 2003. http://www.ece.northwestern.edu/OTC/. (November 2004).

specific, there exist standardised procedures which help to formulate them, like the algebraic method of Gomory. *Branch and cut* techniques combine branch and bound and the use of cutting planes. *Column generation* techniques are used when the linear programming formulation has a huge number of variables. In this case the trick is solving the primal and dual of a linear programming model restricted to a subset of the constraints and to enlarge it afterwards if the solution found is not optimal for the original problem. The name of the technique derives by the fact that the linear programming system is restricted to a subset of columns of the matrix elements a_{ij} and columns that have not yet been included are iteratively added. Finally, *branch and price* techniques arise from the combination of branch and bound with column generation. Good branching rules are particularly difficult to devise in branch and price and are, in general, problem specific. For a full coverage of integer programming techniques we refer the reader to Wolsey (1998).

2.3.2. Network Flow and Dynamic Programming

Problems that can be represented as flows of commodities in a network can be solved by fast algorithms that exploit the special structure of the problem itself. *Network flow* programs are very fast and can solve enormous problems efficiently. Standard linear programming software can recognise some of these classes of problems and adopt proper algorithms, but also a lot of free software can be found on the Internet. Some of the most important network flow problems are minimal cost flow, maximal value flow, shortest path, and matching. Some more complex problems, under certain circumstances, may be decomposed into some of these basic problems. We will see in Chapter 4 how a fast algorithm for solving the all-pairs shortest path problem can be used in the context of approximate algorithms for the graph colouring problem and in Chapter 6 how algorithms for the matching problem can be embedded in more complex algorithms for the timetabling problem. For a deeper treatment of network flow algorithms we refer to Ahuja et al. (1993).

Dynamic programming performs an intelligent enumeration of candidate solutions but it does this in a backward form, from the last decision to the first. The combinatorial optimisation problem is decomposed into stages of decisions. Dynamic programming is then based on the principle that if the solution is optimal, the last k decisions in chronological order must also be optimal, since the completion of an optimal sequence of decisions must be optimal. In practice, a recurrence relation is defined and solved that leads backward from one stage to the precedent. Dynamic programming demands ingenuity in finding a good decomposition into stages so that a convenient recursion function can be used.

2.3.3. Search approach

Besides its use in integer programming, *branch and bound* can be adopted as a general context search strategy. It consists of two steps: partitioning the set of candidate solutions into mutually exclusive sets, each represented by a node in a tree (*branching*); and computing a lower bound on the cost of any solution in a given subset (*lower-bounding*).

Clearly, the choice of the branching rule and the algorithm for computing the lower bound depend on the given problem.

A different approach, although always based on explicit search tree, is used by *search techniques*. These techniques are sequential algorithms that "augment" a state description starting from an empty state. A node in the tree corresponds in this case to an incomplete candidate solution and each node is expanded with the nodes that it is possible to reach by means of a defined successor function, *i.e.*, by adding elements to the candidate solution. Leaves correspond to complete candidate solutions. A search strategy decides which node to expand next starting from the root of the tree, *i.e.*, the initial state. The book of Russell and Norvig, 2003 in Chapters 3 to 5 gives a good introduction to these methods.

The application of search techniques have been particularly well studied on the binary CSP. A *binary CSP* is a CSP (see page 12 for a definition) in which constraints are only defined between pairs of variables. Every CSP can be easily transformed into a binary CSP and similarly many other combinatorial problems. In our study, the binary CSP is important because of its correspondence with the graph colouring problem. A binary CSP can, indeed, be represented by means of a *constraint graph* in which vertices correspond to variables and edges correspond to constraints between variables. Since it will make easier the exposition in Chapter 4 we give here some more details on search techniques for solving the binary CSP that are used also in the context of graph colouring.

Search techniques for binary CSP deal with incomplete solutions, that is, partial assignments of values to variables where some variables are still unassigned. The construction of a complete assignment that satisfies all constraints proceeds by *backtracking*, *i.e.*, a depth-first strategy in the search tree. The performance of backtracking can be considerably enhanced by using problem specific knowledge. We address three main issues in the design of backtracking algorithms and present the rules adopted in the case of binary CSP.

Variable and value ordering. The ordering of variables and the order of the values to assign can have a considerable impact on performance; therefore, strategies have been considered to decide on this order. For the order of variables, two alternatives are usually effective.

- Choosing the most constrained variable, that is, the variable with the fewest legal values. This choice is based on the idea of picking the variable which is most likely to lead soon to a state in which a variable has no legal values, thereby allowing to prune the search tree.
- Choosing the variable with highest degree, that is, the variable involved in the largest number of constraints with unassigned variables. The rationale for this choice is that it tends to reduce the number of values available for the remaining variables thereby reducing the branching of the tree.

Once a variable has been selected, it may be effective to assign the *least-constraining-value*, that is, the value that rules out the fewest choices for unassigned variables, thus leaving the maximal flexibility for the successive variable assignments.

Information propagation. By looking at some of the constraints earlier in the search, or even before the search starts, the search space can be drastically reduced.

The basic form of information propagation is *forward checking*. In forward checking, whenever a variable X receives a value, the domain of each unassigned variable Y,

connected to X in the constraint graph, is restricted by deleting the values which are incompatible with the value chosen for X.

Constraint propagation may go beyond the neighbours of first order, *i.e.*, it may be productive to check the consistency of the values remaining available for the variable Y by checking in its neighbourhood and prune value assignments to X if they lead later to the impossibility of assigning any value to some other variable.

One has to control, however, that the time spent for propagating constraints does not grow more than the time necessary to doing a simple search.

Intelligent backtracking. Sometimes backtracking to the shallowest node, that is, the most recent decision point, may not be the best choice. It can indeed happen that the cause for failure, that is, reaching a state in which a variable has no legal value, is not due to the most recently assigned variable but to some other one earlier in the branch. Some more intelligent approaches to backtracking can therefore be applied.

All these search techniques for binary CSP are further exploited by constraint logic programming, which includes higher-order constraints and thus enlarges the class of handled constraints. Constraint logic programming is also a well defined language to express the constraints.[9] An introduction to this technique is given by Marriott and Stuckey (1998), while papers on successful applications to real life optimisation problems appear regularly in the International Conference on Principle and Practice of Constraints Programming.

2.4. Stochastic local search methods

Stochastic local search methods are approximate algorithms that not necessarily find optimal solutions. As opposed to approximation algorithms they do not provide provable solution quality and provable run time bounds. In practice, however, they perform often better than any other method, finding good solutions in reasonably short time. Given the difficulty to characterise them analytically, their assessment relies on empirical analysis.

SLS methods are characterised by the use of incomplete knowledge, which makes the search in the search space of the candidate solutions not systematic but rather based on the idea of trial and error. Besides being simple and natural, it is surprising how successful these methods have been proven to be on a variety of problems. They consist essentially in constructing one or more initial candidate solutions and trying to improve them by local changes defined by an opportune neighbourhood relation between solutions. Many widely known and high-performing algorithms based on this paradigm make use of randomised decisions in generating or selecting candidate solutions and are therefore *stochastic algorithms*. The success of stochastic algorithms is based on the adversary argument. Accordingly, an adversary may establish a lower bound for the run time of a deterministic algorithm by constructing an input for which the algorithm performs poorly. The input thus constructed may be different for each deterministic algorithm. A

[9] A popular software system for the development and deployment of constraint programming applications is ECLiPSe. Up-to-date information can be obtained from the ECLiPSe web site. IC-Parc, Imperial College London. "The ECLiPSe Constraint Logic Programming System". 1997. `http://www.icparc.ic.ac.uk/eclipse/`. (February 2005)

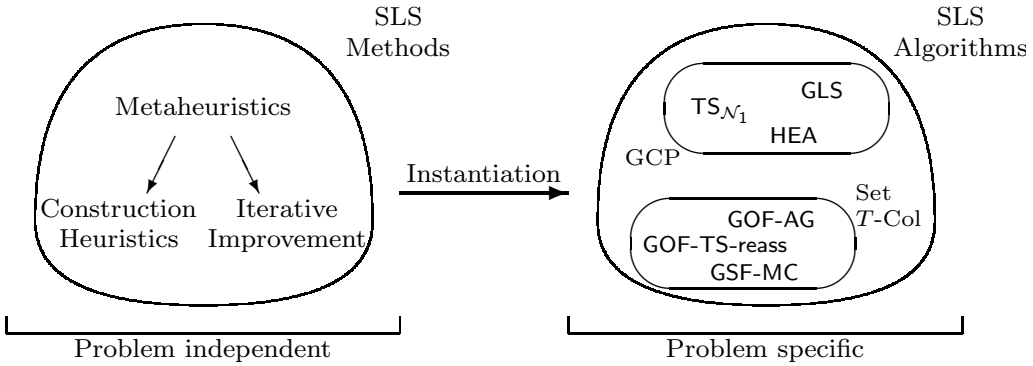


Figure 2.1.: General purpose SLS methods. The instantiation, *i.e.*, configuration, assemblage, and implementation of the components on a specific problem determines the SLS algorithm. The names of the SLS algorithms in the diagram refer to some of the algorithms studied in Chapter 4 and 5.

randomised algorithm can be viewed as a probability distribution on a set of deterministic algorithms. While the adversary may be able to construct an input that foils one (or a small fraction) of deterministic algorithms in the set, it is difficult to devise a single input that is likely to defeat whichever algorithm from the distribution of possible deterministic algorithms implied by a randomised algorithm (Motwani and Raghavan, 1995). In practice, this entails that if we are not satisfied with the result of a run of an algorithm, then we can restart it, as there will be a high chance that the result will be different. From a different perspective, we may assume a distribution of instances and look for the algorithm that performs better on average on this distribution. Then, the use of random choices in algorithms based on the principle of trial and error is introduced to simplify the algorithm from all those deterministic rules that have no positive impact on the final average performance of the algorithm. In practice, in the design of SLS algorithms, a random decision may be invoked whenever no heuristic rule can be conceived that yields a provable improvement in the results of the algorithms on the considered class of instances.

In SLS methods, we can distinguish three classes of components: construction heuristics, iterative improvement, and metaheuristics. The first constructs a solution from scratch, the second improves the solution by performing local changes on it, and the third are high level search strategies. A *stochastic local search algorithm* is the instantiation of an SLS method on a specific problem, through configuration, assemblage, and implementation of its components. Such an organisational scheme is given in Figure 2.1. Whereas construction heuristics and iterative improvement are strongly problem dependent in their instantiation, metaheuristics remain sufficiently problem independent.

According to the definition of Hoos and Stützle (2004), stochastic local search algorithms define a search space and move from some present location to a neighbouring location in this search space. Each location has only a relatively small number of neighbours and *each move is determined by a decision based on local knowledge*. The candidate solutions in the search space may be incomplete (partial) solutions or complete solutions of the given problem instance. SLS methods start with an empty candidate solution and use construction heuristic algorithms to expand incomplete candidate solutions by adding elements according to greedy choices. Once a complete candidate solution has been obtained, other heuristic algorithms, typically called *local search algorithms*,[10] that

[10]Local search algorithms which work with complete candidate solutions are occasionally also referred to

work on complete candidate solutions, change these solutions by modifying one or more of the solution components and thus moving to a new complete candidate solution. More in general, these two phases may not be distinguished and local search may as well work on incomplete candidate solutions. In this case, incomplete solutions are handled not only by adding elements but also by removing or changing elements, and in this local search on incomplete solutions differs from simple construction heuristics.

The first step in applying a local search procedure is the definition of a neighbourhood structure on the set of candidate solutions. The neighbourhood structure defines for each candidate solution the set of possible solutions to which the local search algorithm can move in one step. In its most basic version, called *iterative improvement*, a local search algorithm searches for an improved candidate solution in the neighbourhood of the current candidate solution; if an improved candidate solution is found, it replaces the current solution and the local search is continued until no improving neighbouring solution is found, that is, the algorithm terminates in a local optimum. A disadvantage of iterative improvement is that the algorithm may stop at very poor-quality local optima. An obvious extension would be to restart the algorithm several times from new starting solutions but, in practice, it has been noted that this does not produce significant improvements (Johnson and McGeoch, 1997; Schreiber and Martin, 1999). Several different strategies, more or less problem dependent, have then been proposed to improve basic iterative improvement algorithms and to reach better quality solutions. Some of these strategies are commonly referred to as *metaheuristics* and can be seen as problem independent, general guidance criteria that make use of construction heuristics and iterative improvement algorithms.

In the remainder of this section, we describe the techniques that usually compose a stochastic local search algorithm. We proceed from construction heuristics, to simple local search techniques, and finally to metaheuristics. The field of research is rapidly growing and we do not aim to provide here a comprehensive coverage of the subject. For this, the reader may consult recently published books as those of Aarts and Lenstra (1997), Glover and Kochenberger (2002b), and Hoos and Stützle (2004).

2.4.1. Construction heuristics

Construction algorithms build a solution to a combinatorial optimisation problem in an incremental way. They add solution components step by step and without backtracking, until a complete solution is generated, as shown in Algorithm 2.1. Construction algorithms can be described using the search tree representation adopted for search algorithms (see page 16). Typically, some kind of heuristic rule is employed to add solution components, *i.e.*, to decide to which child node to move in the search tree. Differently from search algorithms, construction algorithms do not traverse the search space in a systematic manner but limit themselves to a single "promising" descent in the tree. They do not provide, therefore, any guarantee to find the optimal solution and are, hence, *incomplete*.

Greedy construction heuristics add a solution component at each construction step with maximal myopic benefit as estimated by a *heuristic function*, which depends uniquely on the incomplete state represented by the current node of the search tree. Usually, such a procedure ends up in poor quality solutions and more sophisticated rules are needed.

as *perturbative local search algorithms* or *neighbourhood search algorithms*.

Function Construction_Heuristic(problem instance I);
while (s is not a complete solution) **do**
    Add an element to the solution s according to some criterion;
end
return A complete candidate solution $s \in \mathcal{S}$.

Algorithm 2.1: Outline of a general Construction Heuristic algorithm.

One possible method to improve greedy algorithms is to evaluate nodes by combining the cost of reaching the node and the estimated cost of the cheapest path from the node to a complete state. Thus, solution components are added according to the estimated cost to find the cheapest solution through a defined node. In *roll-out* (Bertsekas et al., 1997, also known as pilot method, Duin and Voß, 1999), at each step the cost estimate for all possible branches is computed and the node leading to the best branch is chosen. The cost for each branch is estimated using a greedy heuristic for determining the most convenient path. The same procedure is then repeated at the next node.

2.4.2. Iterative improvement

Four problem specific components must be defined for applying iterative improvement. The first component is the set $\mathcal{S}$ of *candidate solutions*. The second component is an *initialisation procedure* that selects, from the set $\mathcal{S}$, a starting solution. The third component is a *neighbourhood structure*, that is, a mapping $\mathcal{N} : \mathcal{S} \mapsto 2^{\mathcal{S}}$. The neighbourhood of a candidate solution s, denoted as $\mathcal{N}(s)$, is the set of all candidate solutions that are neighbours of s. Note that $\mathcal{S}$ and $\mathcal{N}$ define a graph, called *neighbourhood graph*, where the elements of $\mathcal{S}$ are the vertices and the neighbourhood relationship of $\mathcal{N}$ determines the edges between the vertices.[11] The fourth, and last, component of local search, is the *evaluation function*, $f : \mathcal{S} \to \mathbb{R}$, necessary to guide the search through $\mathcal{S}$. The evaluation function serves to assess candidate solutions in the neighbourhood of the current solution and thus to gain the local information necessary to decide where to move. Often, the objective function characterising the optimisation problem is used as the evaluation function, that is, $f \equiv g$. Commonly, but not necessarily, a global optimum for the evaluation function corresponds to an optimal solution of the problem.

These components lead to introduce the concept of local optimum.

Definition 2.1 *A* local optimum *of an evaluation function f with respect to a neighbourhood structure $\mathcal{N}$ is a candidate solution $s \in \mathcal{S}$ such that $f(s) \leq f(s'), \forall s' \in \mathcal{N}(s)$.*

In addition to these components, a further element to be devised is the *search strategy*. In the most general case, a search strategy is defined by the *step function*, that is, a pair $(s, s') \in \mathcal{S} \times \mathcal{S}$ of neighbouring search positions ($s' \in \mathcal{N}(s)$) such that the probability of the algorithm to go from s to s' is larger than zero. The execution of the step function defines a *move*. The most widely used search strategies are best improvement and first improvement. In a *best improvement* strategy, one of the neighbouring candidate solutions that achieves a maximal improvement in the evaluation function is randomly selected. This requires an exhaustive exploration of the neighbourhood at each step, that

[11] Alternatively, a neighbourhood structure on $\mathcal{S}$ can be defined as a function $\mu : \mathcal{S} \times \mathcal{S} \to \{0, 1\}$. Two solutions $s_1, s_2 \in \mathcal{S}$ are then called neighbours if $\mu(s_1, s_2) = 1$. Often, the function μ is symmetric.

Function Iterative_Improvement(problem instance I);
Generate an initial solution $s \in \mathcal{S}$;
while (s is not local optimum in $\mathcal{N}(s)$) **do**
  Select a solution s' from $\mathcal{N}(s)$ such that $f(s') < f(s)$;
  $s \leftarrow s'$;
end
return A solution s that is local optimum in $\mathcal{N}(s)$.

Algorithm 2.2: Outline of a general Iterative Improvement algorithm.

is, all neighbours are evaluated before selection. In a *first improvement* strategy, instead, the first improving step encountered in the exploration of the neighbourhood is selected. The exploration can be random or ordered. Search steps are often computed faster in first improvement but the improvement is typically smaller than with best improvement. The choice between the two strategies should take into account this trade off. In general however, for large neighbourhoods best improvement becomes costly and first improvement is preferred. Nevertheless, in large neighbourhoods best improvement can be combined with pruning techniques to restrict the neighbourhood by avoiding to examine neighbours which are unlikely, or, provably, cannot, provide any improvement in the evaluation function value. Furthermore, often, speed up programming tricks are possible from which mainly best improvement profits.

In Algorithm 2.2, we outline an iterative improvement procedure as it arises from the components defined above. The initial solution is commonly created by a construction heuristic. Depending on alternative choices for each component, despite its simplicity iterative improvement exhibits already a large number of different possible configurations which must be studied for each specific problem.

Intuitively, the larger the set of solutions in the neighbourhood structure is the larger the improvement achievable should be. In this thesis we denote as *very large-scale neighbourhood* (VLSN) a neighbourhood $\mathcal{N}(s)$ that is too large to be explored explicitly. The concept of VLSN is related to the size of the neighbourhood defined as the number of its elements, $|\mathcal{N}(s)|$. VLSN are all neighbourhoods whose size grows exponentially with the size of the instance. Also a neighbourhood with a size that grows as $\mathcal{O}(n^3)$ may as well be too large to be searched effectively in practice if n is large and if the evaluation of a neighbour is costly. The size of the neighbourhood determines a trade off between the quality of solutions and the cost of exploration and its choice must be carefully evaluated both in relation with the problem to be solved and the higher level criteria which guide the local search.[12]

Large scale neighbourhood structures may also be obtained by combining smaller structures:

Definition 2.2 *Given a candidate solution $s \in \mathcal{S}$ and k neighbourhood structures $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_k$ defined on $\mathcal{S}$, the* union of the k neighbourhood structures, *denoted as $\mathcal{N}_1(s) \cup \mathcal{N}_2(s) \cup \cdots \cup \mathcal{N}_k(s)$ consists of the set of candidate solutions s' which belong to at least one of the k neighbourhood structures.*

Definition 2.3 *Given a candidate solution $s \in \mathcal{S}$ and k neighbourhood structures $\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_k$ defined on $\mathcal{S}$, the* composition of the k neighbourhood structures, *denoted as $\mathcal{N}_1(s) \circ$*

[12] Clearly, if the neighbourhood coincides with the whole search space, *i.e.*, $\mathcal{N}(s) = \mathcal{S}$, finding the best solution in $\mathcal{N}(s)$ corresponds to finding the optimal solution; then, if the problem is $\mathcal{NP}$-hard, obviously, finding the best solution in this neighbourhood is also an $\mathcal{NP}$-hard problem.

$\mathcal{N}_2(s) \circ \cdots \circ \mathcal{N}_k(s)$, *consists of the set (chain) of candidate solutions* s' *reachable in an ordered sequence of moves through the* k *neighbourhoods, that is,* $s' \in \mathcal{N}_k(s_{k-1}), s_{k-1} \in \mathcal{N}_{k-1}(s_{k-2}), \ldots, s_1 \in \mathcal{N}_1(s)$.

Considering the union of neighbourhoods is rarely a good choice in practice. A more successful approach is to change the neighbourhood during the search. In *variable neighbourhood descent* (VND), the k neighbourhoods are ordered, typically, according to increasing size. The algorithm starts with neighbourhood $\mathcal{N}_1$ and performs iterative improvement steps until a local optimum is reached. Whenever no further improvement is found for a neighbourhood $\mathcal{N}_i$ and $i+1 \leq k$, VND continues the search in neighbourhood $\mathcal{N}_{i+1}$; if an improvement is obtained in $\mathcal{N}_i$, the search process switches back to $\mathcal{N}_1$: these steps are repeated until no improvement can be found in any of the k neighbourhoods. The general idea of changing the neighbourhood during the search belongs originally to a method called *variable neighbourhood search*, that was introduced by Hansen and Mladenovic (2002).

The neighbourhood composition grows in size even faster than the union and, therefore, it is also rarely a good choice. *Variable depth search* and *ejection chains* are techniques that compose search steps using heuristic rules in order to drastically reduce the time for the examination of the composed neighbourhood. Variable depth search methods are state-of-the-art algorithms for the travelling salesman problem (Lin and Kernighan, 1973), the graph partitioning problem (Kernighan and Lin, 1970), and several others. The methods use cost and tabu restrictions in the selection of the constituting search steps and alternate steps that lead to feasible and infeasible solutions, respectively. Ejection chains were introduced by Glover (1996) as a structured super-class of variable depth methods. Subsequently, their use was restricted to sequences of successive steps each influenced by the precedent and determined only by myopic choices. It is worthwhile to remark, however, that both variable depth search and ejection chains techniques do not guarantee the local optimality of the solutions produced with respect to basic moves. Another approach dealing with complex search steps is *dynasearch*, in which a number of mutually independent steps are executed in parallel. Finally, Di Gaspero and Schaerf (2003b) describe a multi-neighbourhood framework to incorporate several possibilities to take advantage of a multi-neighbourhood structure definition.

2.4.3. Metaheuristics

More sophisticated SLS algorithms are obtained by the usage of *metaheuristics*[13]. These are general-purpose algorithmic concepts that are used to guide underlying problem specific heuristics, such as construction heuristics or iterative improvement algorithms. In this thesis, we regard metaheuristics as problem independent strategies whose application to a specific problem contributes to the complete definition of an SLS algorithm. A minimal requirement for a metaheuristic is to improve over a simple random restart iterative improvement or construction heuristic by making a more intelligent use of the

[13]The prefix *meta* [Gr. at a higher level] to the term *heuristic* [Gr. to find out, discover] is meant to denote another superimposed heuristic which deals with ulterior issues related with the original heuristic. It was introduced in optimisation by Glover (1986). Nothing depends on names, of course. Here, contrary to some other works in the literature, we adopt the term metaheuristic solely to indicate a general guidance criterion over a lower level heuristic or local search. As such it is a method and not a finite algorithm. In our nomenclature, the instantiated algorithm that uses a metaheuristic is called Stochastic Local Search algorithm.

computation time available. In most cases, metaheuristics are based on appropriate common sense rules to improve the local optimum provided by the underlying heuristics.

With the introduction of metaheuristics, SLS algorithms do not have anymore a natural termination condition. Contrary to construction heuristics that end when a complete solution has been constructed or to iterative improvement that end when a local optimum has been reached, metaheuristics use arbitrary run time to search for better solutions. In principle, well designed SLS algorithms should allow to reach better solutions the longer computation times are. Yet, since unlimited computation time is not a realistic assumption, the following termination conditions are usually adopted: (i) a maximal CPU time; (ii) a maximal number of search steps; (iii) a maximal number of search steps without improvement (*idle iterations*). Alternatively, if an optimal solution is known *a priori* then the procedure may end when the optimal solution has been reached (*e.g.*, in the CSP, when all constraints are satisfied).

Several ways to classify metaheuristics have been proposed: population-based versus single search point; dynamic versus static objective function; nature-inspired versus non-nature inspired; one versus various neighbourhood structures; memory usage versus memory-less methods (see also Stützle, 1998). In our presentation, we proceed from metaheuristics that work on single solutions (also called *trajectory methods* as they perform steps in a finite sequence of search positions), over more complex methods working on a population of solutions, to hybrid methods.

Randomised Iterative Improvement

The first idea to make iterative improvement proceed the search beyond local optima is to accept side walk steps. *Side walk steps* are moves that lead to candidate solutions whose evaluation function is the same as the current solution. A further possibility is to accept also worsening candidate solutions. The problem in doing this is that if worsening solutions are accepted deterministically, the risk is to undo the step immediately after leading to a possible cycling behaviour.

A way to avoid this drawback is to determine when a worsening step has to be performed by using a probabilistic criterion. The simplest algorithm which does this is *Randomised Iterative Improvement* (RII). In RII, a parameter $wp \in [0, 1]$, called walk probability, is used to probabilistically determine whether a worsening step or an improving step has to be performed. How this happens is shown in Algorithm 2.3. At each step a random number uniformly distributed in the interval $[0, 1]$ is drawn. If the number is less or equal wp, a candidate solution from the neighbourhood $\mathcal{N}(s)$ is chosen randomly, otherwise an improving candidate solution as in a best improvement or a first improvement strategy is taken.

Simulated Annealing

Simulated Annealing (SA) (Černý, 1985; Kirkpatrick et al., 1983) is inspired by statistical mechanics, a discipline of Physics for analysing the property of atoms in solid matter. In statistical mechanics each configuration of positions of a collection of atoms is weighted by a Boltzman probability factor that depends on the energy of the configuration and on the temperature. The fundamental issue concerns what happens to the system of atoms when the temperature is decreased. The configurations ideally collapse into those of

Function Randomised_Iterative_Improvement(problem instance I, walk probability wp);
Generate an initial solution $s \in \mathcal{S}$;
$s_{best} \leftarrow s$;
while (termination condition not satisfied) **do**
  $p \leftarrow random([0,1])$;
  if ($p \leq wp$) **then**
    Select a solution s' from $\mathcal{N}(s)$ randomly;
  else
    Select a solution s' from $\mathcal{N}(s)$ such that $f(s') < f(s)$;
  end
  if $f(s) < f(s_{best})$ **then** $s_{best} \leftarrow s$;
  $s \leftarrow s'$;
end
return s_{best}

Algorithm 2.3: Outline of a general Randomised Iterative Improvement algorithm.

Function Simulated_Annealing(problem instance I, initial temperature T);
Generate an initial solution $s \in \mathcal{S}$;
$s_{best} \leftarrow s$;
while (termination condition not satisfied) **do**
  Select a solution s' from $\mathcal{N}(s)$;
  $s \leftarrow s'$ with probability $p_{accept}(T, s, s')$;
  if $f(s) < f(s_{best})$ **then** $s_{best} \leftarrow s$;
  Update temperature T;
end
return s_{best}

Algorithm 2.4: Outline of a general Simulated Annealing algorithm.

perfect lattice structure corresponding to the states of minimal energy.

Metropolis et al. (1953) introduced an algorithm that provides an efficient simulation of a collection of atoms in equilibrium at a given temperature. At each step a random perturbation to the current configuration is generated and the probability of accepting a configuration of higher energy is given by the Boltzman probability distribution. Analogously, in combinatorial optimisation a randomly chosen solution is accepted according to the deterioration that it brings in the evaluation function value, such that the more a candidate solution worsens the evaluation function the less likely it is accepted. The convergence to the solution of the minimal evaluation function value that simulates the annealing process in Physics is determined by a parameter, called *temperature*, which is opportunely varied during the search. The probability function derived from the Metropolis algorithm for matching the needs of combinatorial optimisation is:

$$p_{accept}(T, s, s') = \begin{cases} 1 & \text{if} \quad f(s') \leq f(s) \\ \exp(\frac{f(s)-f(s')}{T}) & \text{otherwise} \end{cases} \tag{2.2}$$

where s is the current solution and $s' \in \mathcal{N}(s)$. The parameter T is the temperature and determines how likely it is to perform worsening steps during the search. The lower T is the less likely a deteriorating configuration is accepted. Algorithm 2.4 outlines the SA procedure.

The temperature is set relatively high at the beginning of the search and is then decreased according to a cooling schedule. The exact characterisation of the temperature over the whole search depends on the implementation. In this thesis we consider the temperature at each step of a Simulated Annealing algorithm determined by the following factors:

- the *initial temperature*, often determined from the average solution quality of a sample of the neighbouring solutions;
- the *temperature length*, which defines the number of search steps to perform at a fixed temperature value; usually it is set proportional to the size of the neighbourhood;
- the *cooling trend*, as determined, for example, by the standard formula $T_{i+1} = \alpha \cdot T_i$, where α is a parameter called *cooling rate* and T_i is the temperature value at the iteration i of the algorithm;
- the *re-heating mechanism*, used to increase the temperature in case no improvement is found, for example, for a number of steps proportional to the temperature length.

A constant temperature over all the search may also provide good results but determining a good value for it seems difficult (Fielding, 2000).

Tabu Search

Another strategy to allow iterative best improvement to perform worsening moves without immediately undoing them in the next step is to remember recently visited solutions and avoid to return to them. *Tabu Search* (TS), introduced independently by Glover (1989), Glover (1990), and Hansen and Jaumard (1990), uses a short-term memory to restrict the neighbourhood of the current solution to a subset $\mathcal{N}'(s) \subseteq \mathcal{N}(s)$ of admissible neighbours. In practice, not the complete candidate solutions are memorised, as this would be costly in terms of space and time for comparisons, but solution components. A parameter tt, called *tabu tenure* determines the duration in terms of search steps in which the re-insertion or removal of these solution components is forbidden. When memorising only solution components, it may happen that some attractive solutions become forbidden. To avoid this, an *aspiration criterion* is commonly introduced which specifies conditions under which the tabu status of a candidate solution may be overridden. See Algorithm 2.5 for an outline of a TS procedure.

Many refinements have been proposed in the literature. Taillard (1991) improves the robustness of performance by choosing tt randomly from an interval $[tt_{min}, tt_{max}]$. Battiti and Tecchiolli (1994) propose a reactive mechanism to dynamically adjust the tabu tenure during the search based on the detection of trajectory repetitions. Finally, Glover and Laguna (1997) report many ideas to include also a long-term memory and extend $\mathcal{N}(s)$ through the inclusion of *elite candidate solutions*. They focus on the trade off between intensification and diversification, *i.e.*, between exploiting the experience accumulated during the search and exploring new, unseen regions of the search space.

Function Tabu_Search(problem instance I, tabu tenure tt);
Generate an initial solution $s \in \mathcal{S}$;
Initialise memory structures;
$s_{best} \leftarrow s$;
while (termination condition not met) **do**
  Let $\mathcal{N}'(s) = \{s' \in \mathcal{N}(s) : s' \text{ is non-tabu or satisfies an aspiration criterion}\}$;
  Let s' be the best solution in $\mathcal{N}'(s)$;
  $s \leftarrow s'$;
  if $f(s') < f(s_{best})$ **then** $s_{best} \leftarrow s$;
  Update memory structures;
end
return s_{best}

Algorithm 2.5: Outline of a general Tabu Search algorithm.

Dynamic Local Search

A different approach for escaping local optima is to modify the evaluation function in such a way that further improvement steps become possible. This can be done by associating penalty weights to individual solution components which have an impact on the objective function. Whenever the iterative improvement ends in a local optimum, the penalties of some solution components present in the solution are changed and the evaluation function is updated with the new weights. This procedure is called *Dynamic Local Search* and corresponds to a repeated iterative improvement procedure with a dynamically changing evaluation function. Worsening steps with respect to the original evaluation function are no longer necessary while side walk steps may be accepted in a limited number according to the search strategy adopted. The modified evaluation function is usually expressed in the form:

$$f'(s) = f(s) + \lambda \cdot \sum_{i=1}^{n} w_i \cdot I_i(s)$$

where w_i is the penalty weight of solution component i and $I_i(s)$ is an indicator function which returns 1 if the solution component i is present in s and 0 otherwise. The parameter λ is used to control the relative weight of the penalties on the evaluation function.

The penalties are initially set to zero and subsequently updated after each new iterative improvement run. The update may involve all or only some of the solution components present in locally optimal solution and variants of dynamic local search may differ in this.

In *Guided Local Search* (GLS), the solution quality contribution of a single solution component, $f_i(s)$, is defined to estimate the utility $util_i$ of increasing the penalty weight of component i:

$$util_i = \frac{f_i(s)}{1 + w_i}.$$

Only solution components with maximal utility values are updated by setting $w_i = w_i + 1$. The rationale behind this mechanism is that solution components with high negative impact in the solution should increase their penalties because of their high cost contribution. The denominator is used to avoid too frequently penalising the same components. In some refined implementations, weights are occasionally smoothed during the search

Function Guided_Local_Search(problem instance I);
Generate an initial solution $s \in \mathcal{S}$;
$s_{best} \leftarrow s$; Initialise penalty weights;
while (termination condition not met) **do**
  Update evaluation function $f' = f + \lambda \cdot \sum_{i=1}^{n} w_i \cdot I_i(s)$;
  $s' \leftarrow$ Iterative_Improvement(s, f');
  if $f(s') < f(s_{best})$ **then** $s_{best} \leftarrow s$;
  $s \leftarrow s'$;
  forall $\{i \in [1, \ldots, n] : \arg\max_i util_i(s)\}$ **do**
    $w_i = w_i + 1$
  end
end
return s_{best}

Algorithm 2.6: Outline of a general Guided local search algorithm.

(Mills and Tsang, 2000). Other updating schemes, studied mainly on the logic satisfiability problem, use probabilistic mechanisms to select the components to penalise (see Hutter et al., 2002 for an example).

Iterated Local Search

In Section 2.4.2, we described the possibility of escaping from local optima by using different neighbourhoods. Based on the same idea, *Iterated Local Search* (ILS) uses an iterated improvement procedure to find local optima, and a *perturbation* to modify the solution in such a way that the local search can be continued and new regions of the search space be explored. Perturbations are composed of one step (or occasionally a number of steps) in another neighbourhood, not necessarily guided by an evaluation function. An iterative improvement procedure is then applied from the perturbed solution such that a new local optimum is found. Finally, an acceptance criterion decides whether to continue the search from the new or from the previous local optimum. Often, a probabilistic criterion is adopted. Algorithm 2.7 outlines a basic ILS procedure. More elaborated implementations may include also a history component, used in the acceptance criterion on the perturbation. However, despite its simplicity, high performing ILS algorithms require problem specific perturbations and devising them may require some effort. For deeper insights we refer to Lourenço et al. (2002).

Adaptive iterated construction search

Greedy construction heuristics are widely adopted to generate the initial solutions of all the local search methods introduced above. Their involvement can be stronger. An appealing idea is to exploit gained experience for guiding new solution constructions. In *Adaptive Iterated Construction Search* (AICS), weights are associated with possible decisions that are made during the construction process. These weights are adapted over multiple iterations of the search process to reflect the experience from the previous iterations. As outlined in Algorithm 2.8, in typical implementations of AICS each iteration consists of three stages: first a construction search process is used to generate a candidate solution s; next, an additional perturbative iterated improvement process is performed

Function Iterated_Local_Search(problem instance I);
Generate an initial solution $s \in \mathcal{S}$;
$s \leftarrow$ Iterative_Improvement(s);
$s_{best} \leftarrow s$;
while (termination condition not met) **do**
  Let s' be the solution obtained by applying a perturbation to s;
  $s'' \leftarrow$ Iterative_Improvement(s');
  if $f(s'') < f(s_{best})$ **then** $s_{best} \leftarrow s''$;
  Accept s' or s'' as the new s;
end
return s_{best}

Algorithm 2.7: Outline of a general Iterated Local Search algorithm.

Function Adaptive_Iterated_Construction_Search(problem instance I);
Initialise weights;
while (termination condition not met) **do**
  Construct a solution $s \in \mathcal{S}$ using a heuristic h and weights;
  $s' \leftarrow$ Iterative_Improvement(s);
  if $f(s') < f(s_{best})$ **then** $s_{best} \leftarrow s'$;
  Update weights according to components and quality of s';
end
return s_{best}

Algorithm 2.8: Outline of a general Adaptive Iterated Construction Search algorithm.

on s, yielding a locally optimal solution s'. Finally weights are adapted based on the solution components used in s' and the solution quality of s'. Squeaky wheel optimisation (Joslin and Clements, 1999) and iterated greedy (Culberson, 1992) are successful AICS methods based on single solutions. Adaptive memory algorithm (Rochat and Taillard, 1995) and Ant Colony Optimisation are AICS methods based, instead, on a population of solutions.

Ant Colony Optimisation

Ant Colony Optimisation (ACO) is a nature-inspired, population-based metaheuristic. ACO mimics the behaviour of ants that indirectly communicate via distributed, dynamically changing information, the so-called *pheromone trails*. In ACO, in each iteration a population of k candidate solutions is generated by a construction procedure that uses probabilistic decisions. Solution construction is biased by the pheromone trails, which are weights that influence decisions in a similar manner as AICS, and heuristic information on the instance being solved. Once the solutions are constructed, typically some or all candidate solutions are improved by an iterative improvement procedure (or a more elaborated local search method), resulting in a population of locally optimal solutions. Finally, the pheromone trails are updated on the basis of the current candidate solutions and their quality. Typically, the update implies first a decrease of all pheromone trails by a constant factor, which corresponds to pheromone evaporation in nature, and then an increase for the subset of pheromone trails determined by the current population and the best solution found so far. The whole procedure is outlined in Algorithm 2.9. The

Function Ant_Colony_Optimisation(problem instance I);
Initialise pheromone;
while (termination condition not met) **do**
  Construct a population of solutions $sp \subseteq S$ exploiting pheromone and heuristic information;
  for all $s \in sp$ **do**
    $s \leftarrow$ Iterative_Improvement(s);
  end
  if $\min_{s \in sp} f(s) < f(s_{best})$ **then** $s_{best} = \arg\min_{s \in sp} f(s)$;
  Update weights according to solution components and quality of sp or s_{best};
end
return s_{best}

Algorithm 2.9: Outline of a general Ant Colony Optimisation algorithm.

first available ACO algorithm is Ant System but it achieved only poor results. A more successful refinement of ACO is, instead, $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System which has a peculiar way to update pheromone trails (Stützle, 1998; Stützle and Hoos, 2000). For a comprehensive coverage of Ant Colony Optimisation we refer the reader to Dorigo and Stützle (2004).

Evolutionary algorithms

Evolutionary algorithms (EA) is a family of approaches inspired by the capability of nature to evolve living beings adapting them to their environment. This paradigm may be used for modelling real evolutionary processes and for optimisation. In EA for combinatorial optimisation, a population of candidate solutions is maintained and a series of genetic operators are repeatedly applied to replace, partially or totally the population with the next one. Typically, two operators are used to modify a solution. A *mutation* operator modifies an individual of the population usually by random changes. A *recombination* operator generates one or more individuals (offspring) by combining information from two or more individuals (parents). The most commonly used type of recombination operator is called *crossover* and consists in assembling pieces from an appropriate representation of two individuals. Finally, a selection criterion chooses the solutions for the next generation based on their fitness (usually associated with the value of the objective function). Individuals with higher fitness have higher probability of being selected thus applying the principle of survival of the fittest. Often preferred in combinatorial optimisation are two variants of EA: *Genetic algorithms* and *Memetic algorithms*. Genetic algorithms use a discrete solution representation based on bit strings of equal length. Memetic algorithms are genetic algorithms in which a local search procedure is applied after the mutation and recombination operators. Memetic algorithms typically achieve higher performance than Genetic algorithms in most combinatorial problems. The outline in Algorithm 2.10 is a basic version of a Memetic algorithm. The recombination operator returns the current population sp enlarged with the offsprings. It implements the inheritance of desirable properties of the parent solution and it is therefore highly problem specific. The selection operator either replaces the whole population (generational approach) or replaces less fit individuals (steady-state approach). Two reference books for EAs in combinatorial problem solving are Holland (1975), Goldberg (1989), and Davis (1991).

Function Memetic_Algorithm(problem instance I);
Generate a population of solutions $sp \subseteq S$;
while (termination condition not met) **do**
  $sp' \leftarrow recombination(sp)$;
  $sp'' \leftarrow mutation(sp' \cup sp)$;
  $sp' \leftarrow$ Iterative_Improvement$(sp' \cup sp'')$;
  if $\min_{s \in sp'} f(s) < f(s_{best})$ **then** $s_{best} = \arg\min_{s \in sp'} f(s)$;
  $sp \leftarrow selection(sp')$;
end
return s_{best}

Algorithm 2.10: Outline of a general Memetic Algorithm.

Other methods

We mention shortly other metaheuristics, which we omitted because they are not used in this thesis. *Greedy randomised adaptive search* procedures are a two phase approach which consist in applying a randomised construction heuristic and then improving the so generated candidate solution by an iterative improvement procedure. *Scatter search* and *path relinking* are two evolutionary algorithms. They use reference solutions spread in the search space according to some diversity measure and then join or recombine some of them with good quality and sufficient diversity. The peculiarity of path relinking is that the trajectory connecting reference solutions are explored by generating paths in the neighbourhood space. *Estimation of distribution algorithms* (Larrañaga and Lozano, 2001) use machine learning principles to solve optimisation problems. They try to learn the locations of the most promising regions of the search space, by constructing a probabilistic model, such as a graphical model, to generate candidate solutions, and learning is used to adapt the probability model to explore more promising regions of space. Bayesian optimisation algorithms (Pelikan et al., 1999) consider multiple solution component dependencies in the probabilistic model. The field of estimation of distribution algorithms is quite young and much research effort has focused on methodological aspects rather than on high-performance applications.

2.4.4. Hybrid methods

Practice showed that high performance algorithms are usually achieved by hybrid algorithms. This is particularly true for real-life problems, where the complexity of the constraints may require several approaches to work together in order to satisfy them all. Talbi (2002) proposes a taxonomy of hybrid metaheuristics and collects references from published literature. Here, we distinguish two main classes of hybrid algorithms relevant to our study: combinations of metaheuristics, that include components from several metaheuristics, and combinations of stochastic local search algorithms with exact algorithms.

Combination of metaheuristics

The most common way of hybridising metaheuristics is represented by embedding trajectory approaches into population based approaches or other search strategies. In all cases, the goal is to find a better trade off between intensification and diversification of the search. Trajectory methods are better in carefully exploring promising areas of the search space, whereas population based methods, and also ILS and AICS, are good in identifying promising areas of the search space. We already described memetic algorithms, which actually are an example of hybridisation of evolutionary algorithms with local search. Other possibilities arise by the substitution of simple iterative improvement procedures in the algorithms outlined above by more complex local searches like variable neighbourhood descent, tabu search, or simulated annealing. In general, the use of more powerful iterative improvement procedures reduces the risk of missing good solutions in a region of the search space before moving away to investigate other regions.

Another approach consists in alternating different strategies during the search. For example one may first use fast improvement algorithms and then slower approaches like Simulated Annealing. In particular, VND and TS are usually good for solving hard constraints while SA helps for optimising soft constraints when long run times are available (in Chapter 6 we give empirical support to this claim in the context of timetabling).

Combination of stochastic local search methods with exact algorithms

The combination of local search methods with exact algorithms is currently receiving particular attention in the research community. The following areas of research, depending on the type of hybridisation, may be characterised.

Type 1. An SLS algorithm and an exact algorithm are used sequentially for solving different subproblems. Usually, an SLS algorithm is used to solve large size problems while subproblems of smaller dimension can be solved by exact methods.

Type 2. Constraint programming techniques are used to reduce the search space to be exploited by local search algorithms.

Type 3. Exact algorithms are used for efficiently exploring large neighbourhoods, instead of simply enumerating or randomly sampling the neighbourhood.

Type 4. Concepts from stochastic local search methods, like tabu rules or probabilistic choices, are adopted into tree search algorithms.

Hybridisations of Type 1 are particularly well suited for problems with a large number of constraints. An example is the travelling salesman problem with time windows (TSPTW), in which each node must be visited in a predefined time interval. An SLS approach is well suited for finding a tour, *i.e.*, a sequence of points, while an exact method may be used to test whether for a created tour there exists an assignment of departure and arrival times such that all time constraints are satisfied. We will see a similar application in the chapter on timetabling. The TSPTW has also been object of applications of hybridisation of Type 2. In this case, constraint programming is used to reduce the neighbourhood to only feasible solutions, such that the solutions considered are only those that satisfy time constraints (Pesant and Gendreau, 1996). Ahuja et al. (2002) review many possibilities which arise with hybridisation of Type 3. They focus on very large-scale

neighbourhoods for partitioning problems which can be efficiently searched by network flow algorithms. Lourenço (1995) presents an example where an exact method is used to define the perturbation in ILS by solving a simplified subproblem. The same idea is purported and generalised by Dumitrescu and Stützle (2003). Particularly successful applications of hybrid algorithms of Type 4 are tree search with probabilistic backtracking (instead of deterministic node backtracking). In order to search exhaustively some basic bookkeeping procedures similar to tabu rules may be adopted. Gomes and Shmoys (2004) and Prestwich (2002b) apply these techniques to the constraint satisfaction problem and to graph colouring.

Besides these types, the combination of local search methods with integer programming is also currently investigated (Dumitrescu and Stützle, 2003). An example in this direction are local branching techniques (Fischetti and Lodi, 2003). However, very little has been done in this area. Finally, exact methods may be used to determine lower bounds for a problem (an application will be shown in Chapter 5).

2.4.5. Theoretical remarks

The attempt to determine the algorithm that works the best for all the problems is doomed to fail. This fact is investigated from a theoretical point of view and results are collected under the name of *no free lunch theorems* for combinatorial optimisation (Wolpert and Macready, 1997). In short, the *no free lunch theorem* states that all search algorithms have the same average performance over all possible objective functions $f : \mathcal{X} \rightarrow \mathcal{Y}$, where the search space $\mathcal{X}$ as well as the cost function $\mathcal{Y}$ are finite sets.[14] This fact has been intended as an argument in favour of the *specialisation* of algorithms: one does not need an algorithm that performs well on all possible functions, but only on a subset that arises from the constraints of the real-life problem. It is, indeed accepted that, in order to make some progress in the search, there must be some regularity in the function and the *no free lunch theorem* tells us that there is no reason to believe that an algorithm would do better than a random search unless the operators of the algorithm are correlated to those regularities. Recently, research has focused on the problem of finding the classes of functions where the *no free lunch theorem* does not hold. In particular, one important issue is whether the *no free lunch theorem* applies to instances of a specific combinatorial optimisation problem, as for example, different types of graphs in the graph colouring problem. Schumacher et al. (2001) showed that the results of the *no free lunch theorem* hold only for the subset of functions which are closed under permutation while Igel and Toussaint (2003) investigate the implications of this result on neighbourhood methods for combinatorial optimisation problems. However, these results still do not prove whether instance types have also a strong influence on algorithm performance. Empirical results suggest that the answer to this issue must be affirmative.

Other theoretical results prove the asymptotic convergence to the global optimum of some stochastic local search methods. Proofs under specific conditions exist for simulated annealing (Hajek, 1988; Faigle and Kern, 1991), for some probabilistic and deterministic tabu search algorithms (Glover and Hanafi, 2002; Faigle and Kern, 1992), for genetic algorithms (Rudolph, 1994), and for specific ant colony optimisation algorithms (Gutjahr, 2002; Stützle and Dorigo, 2002). However, these results have no significant impact in

[14] We do not use the notation $\mathcal{S}$ and $\mathbb{R}$ for $\mathcal{X}$ and $\mathcal{Y}$ because the no free lunch theorem applies to more abstract definitions.

practice.

More interesting, from a practical point of view, may be worst case results. Examples of this type are mainly concerned with the maximal satisfiability problem in logic. Alimonti (1996) and Khanna et al. (1998) developed and studied local search algorithms with approximation guarantees on a generalised version of this problem. Mastrolilli and Gambardella (2004) extend worst case studies to tabu search algorithms. Alternative and complementary to approximation analysis is *domination analysis* that determines for a given combinatorial problem the fraction of solutions that are not better than the solutions found by a heuristic algorithm (Gutin et al., 2003). The main application of this analysis concerned heuristics for the travelling salesman problem (Gutin et al., 2002).

Finally, a theory of complexity exists also for local search algorithms. The class $\mathcal{PLS}$ for polynomial time local search consists of those problems and neighbourhood structures in which (i) some initial solution can be produced in polynomial time, (ii) the cost of the solution can be computed in polynomial time and (iii) determining whether the solution is locally optimal or, if not, generating an improved neighbouring solution can be done in polynomial time. Note that in this definition the number of steps needed to reach a local optimum is not mentioned, which in most cases cannot be bounded by a polynomial. Similarly to the case of search problems and to the classes $\mathcal{P}$ and $\mathcal{NP}$, the concept of $\mathcal{PLS}$-complete problems is introduced to denote the problems for which no local search is known that finds a local optimum in a polynomial number of steps and to which all other problems can be polynomially reduced. Showing that for a $\mathcal{PLS}$-complete problem there exists a local search that finds local optima in polynomial time would imply that finding a local optimum in polynomial time can be accomplished for all problems in $\mathcal{PLS}$ (Johnson et al., 1988). Few problems and neighbourhood structures that are of interest for practical reasons have been shown to be $\mathcal{PLS}$-complete. Among them are the TSP with the Lin-Kernighan algorithm and the partitioning of a weighted graph with the Kernighan-Lin algorithm, the probably two most famous variable depth algorithms.

2.5. Discussion

We introduced combinatorial optimisation problems and we focused on common methods for solving them. These solution methods provide frameworks to look at the problem under specific formalisms. In principle they can be applied to any situation. However, expertise in only one single method can lead to poor algorithms for a specific problem. Problems should be analysed from the point of view of different solution methods outside of any predetermined framework. The task of a researcher in this field is, then, to unveil which are the best choices for problems that are likely to be of wide application. For real-world problems with many constraints, which are relevant only for contextual applications, the task of understanding the best solution approach is left to the practitioner. Nevertheless, as we will see, the adoption of a systematic engineering methodology can bring big advantages in this context.

In the case of stochastic local search methods, we saw that several components must be defined. Techniques for determining an initial solution, solution representation, evaluation function, neighbourhood structures, etc. cannot be standardised, but require to be specifically investigated for the problem at hand.

Crucial for the success of an SLS algorithm is the definition of the neighbourhood structure. Its definition and analysis will receive central attention throughout the whole the-

sis. In particular, we are interested in the assessment of very large-scale neighbourhoods which can be searched effectively. Ahuja et al. (2002) and Thompson and Orlin (1989) showed that for problems that can be formalised as partitioning problems, it is possible to design a cyclic or path exchange neighbourhood which is searched effectively by a dynamic programming approach. Graph colouring has indeed the propriety of being a partitioning problem. Cyclic and path exchange neighbourhoods, as conceived by Ahuja et al. (2002), appear even more powerful than variable depth methods because the neighbourhood may be searched exhaustively. Previous results with VLSN appear to be encouraging and in some cases new algorithms with best peak performances are obtained. A list of published works on different problems is maintained online by Ahuja and Orlin.[15] As a consequence of this arising interest in VLSN, its application and assessment on the graph colouring problem is of timely interest.

Another interesting aspect of SLS algorithms is the possibility of easily assembling different algorithms thus obtaining hybrid methods. Usually, hybrid algorithms outperform basic implementations and since the scope of this thesis is studying high-performance algorithms for the mentioned problems, in practice we end up studying hybrid algorithms. Almost all the typologies of hybridisations introduced in Section 2.4.4 will be applied. Providing a methodology for their development which is likely to work also for different problems from those studied here is a hard task. An attempt in this direction will be given in Chapter 6, where we try to outline some guidelines for the application of hybrid SLS techniques that imply the combination of several components and metaheuristics. The hybrid algorithm, which is selected for solving the timetabling problem, may also be very inspiring for applications to different problems. Support to this claim is provided by the fact that the final algorithm designed by a team of researchers, including the author, for solving the car sequencing problem proposed in the context of the ROADEF 2005 competition ended up being very similar to the one reported here, although the development procedure was completely different (Risler et al., 2004). However, we are convinced that the only correct procedure for the development of hybrid SLS algorithms is the adoption of a systematic experimental methodology for iterative refinements of the algorithm.

There are, of course, several other important elements of combinatorial optimisation which remain marginal to our discussion. One of such aspects are parallel algorithms. Computer industry is shifting towards parallel processors machines[16] and research has been recently focusing on systematic analysis of parallel computation algorithms. In the field of SLS methods, Crainic et al. (1997) propose a taxonomy of parallel metaheuristics and present methodological lines and possible future research directions.

Another recent field of research, which requires separate theoretical modelling, is *online* optimisation. In many practical situations not all input data that define a problem instance are available in advance and decisions have to be made based on incomplete knowledge. In an online optimisation problem, the input is modelled as a (finite) sequence of requests which are supplied to the algorithm incrementally. Then, an online algorithm produces the output incrementally without knowing the complete input. An alternative way to deal with *online* optimisation is *stochastic* optimisation which assumes probability distributions of the inputs and search for the best solution on average (see Bianchi et al., 2004 for a case study). Stochastic optimisation is also strictly connected

[15] R.K. Ahuja and J.B. Orlin. *VLSN Papers*. March 2005. http://www.ise.ufl.edu/ahuja/vlsn/index.htm. May 2005.

[16] It has been recently announced by Intel and AMD that the next step to enhance their respective Pentium 4 and Opteron processors will be the release of dual-core processors, that is, chips that contain two processors both running at the same frequency.

with *programming under uncertainty* which is of main interest above all in financial applications. In this thesis, we confine ourselves to consider static and deterministic problems, in which all the data of the instance are completely known in advance.

Chapter 3.

Statistical Methods for the Analysis of Stochastic Optimisers

In which we define the experimental designs that are of interest for the assessment of algorithms and define the appropriate statistical methods for their analysis.

3.1. Introduction

Stochastic local search methods perform well in practice. Yet, their theoretical characterisation is hard and, often, with scarce practical impact. The assessment of SLS algorithms is therefore carried out through empirical analyses. The central issue of these analyses is the comparison of algorithms among each other and against benchmark algorithms that constitute the state-of-the-art for a given problem. In this chapter, we define the statistical methodologies for carrying out such empirical comparisons of stochastic optimisers in a systematic and correct way. Statistical methods have only been recently recognised as appropriate procedures for the assessment of algorithm performance and a clear methodology has not yet been established. Our contribution is the identification of the test procedures which may be applied in different possible scenarios of analysis. Parametric statistics seems inappropriate for describing algorithm performance because some of its assumptions are very likely to be violated in this context. We focus, therefore, also on alternative methods from non-parametric statistics. In particular, we define permutation tests for experimental designs that are of interest in the comparison of algorithms and we provide the algorithmic procedure for their implementation. Some of these procedures are a novel contribution of this thesis. In addition to this, we develop a visualisation of results for multiple comparisons by means of simultaneous confidence intervals which allows for an immediate visual perception of descriptive as well as inferential statistics. This representation, inspired by applications of parametric statistics, is here firstly suggested. The implementation of permutation tests is a current active field of research in applied statistics and their introduction and application in the analysis of algorithm performance is new.

The statistical tests thus defined may then be used in sequential testing as a way to reduce the computational efforts of large scale experiments. Sequential tests constitute a particularly helpful tool in the development of SLS algorithms when *a priori* no knowledge is given on which approaches are preferable and a high number of valid alternatives arises. The results of the experiments guide the development of SLS algorithms and we call this process SLS algorithm engineering.

In the Chapter, we also discuss other empirical methods which are useful for gaining deeper insights on the behaviour of specific SLS algorithms. These are the representation of algorithm behaviour over time and the characterisation of problem search space landscape.

3.2. The need for the empirical approach

There are two ways to study the performance of algorithms: one is analytical and relies on mathematical proofs and the investigation of combinatorial properties, the other is experimental and relies on empirical observations. It has been for a long time debated whether empirical methods to analyse algorithms are appropriate. Two main criticisms have been put forward against them. The first is that data structures, coding style, compiler, and machine have an influence on the experimental performance of an algorithm. Hence, algorithm implementations are irreproducible and one cannot establish that an algorithm is efficient or inefficient independently of the apparatus used to test it. The second critic is that in fact machines execute only deterministic procedures and, therefore, it should be possible to deduce their properties in a formal way without need for empirical tests.

Hooker (1996) uses the analogy with physics to reply to these critics. In physics, on the one hand, the principle of uncertainty states that it is impossible to observe the microscopic world without having an influence on it. Thus, similarly, a precise description of complex algorithms without influence of implementation details is likely to be impossible. Nevertheless, this should not discourage us from performing practical tests and relying on their results. The focus must be on a level of analysis which has practical relevance. On the other hand, a mathematical deductive method could be adopted to explain physical phenomena but the complexity of the systems analysed makes it often impossible for a human mind to yield a theoretical characterisation and empirical methods are used, therefore, to suggest and verify hypothesis. In a similar manner, complex algorithms with heuristic choices and long procedures are very hard to be formally understood and empirical testing is needed for gaining insights of practical relevance.

The analogy with physics can be further extended by considering that both the theory of relativity, which explains macroscopic phenomena, and the quantum mechanics, which explains microscopic phenomena, admit the classical physics as a good approximation for problem sizes of daily experience. Similarly, although a precise characterisation of algorithmic operations is possible, it can be limited to a low level of analysis while for problems of larger scale an approximation can be perfectly sufficient for practical scopes.

Moret (2002) gives further credit to the empirical analysis of algorithms arguing that theoretical analysis alone is not enough. For example, the $\mathcal{O}(\cdot)$-notation simplifies the analysis to make results well understood and machine independent. Nevertheless, a lower asymptotic running time may imply better performance only on instance sizes much larger than those that we want to solve. In Chapter 6, we will meet one such case where an exact algorithm that runs in $\mathcal{O}(|V||C|)$ is preferred over one in $\mathcal{O}(\sqrt{|V|}|C|)$ because the sizes of the instances to solve are small and the first algorithm is empirically faster on those sizes of interest. In other cases, the constants that are hidden from the $\mathcal{O}$-notation may as well have a large impact.

Similarly, the results on the worst-case behaviour of an algorithm may be restricted to

a very small subset of instances. An example is the simplex method whose behaviour, despite its exponential worst case, is typically a low-degree polynomial. Approximation algorithms with performance guarantees may present a similar pattern: the results they find may be much better than their worst case guarantees. Empirical studies are therefore needed even for algorithms with alleged theoretical results. A recent example with approximation algorithms for scheduling problems is given by Savelsbergh et al. (2005).

All these elements pushed the algorithmic community to include implementation and testing as an integral part of algorithm development and to recognise empirical tests as equally important as theoretical analysis. In recent years, several papers have tried to set out guidelines for empirical research on algorithms (Barr, Golden, Kelly, Resende, and Stewart, 1995; Pardalos and Romeijn, 2002; Rardin and Uzsoy, 2001; Cohen, 1995; Johnson, 2002; Moret, 2002) and two related areas of research called Experimental Algorithmics and Algorithm Engineering have been established (Fleischer, Moret, and Schmidt, 2002).

All these arguments acquire even more emphasis when we are concerned with stochastic local search algorithms. The stochastic character of these algorithms entails the impossibility to characterise deterministically the whole procedure. In practice, machines cannot work randomly, but procedures that generate random numbers use sequences of numbers such that foreseeing the successor is very hard. Theoretical analyses in this case must rely on probabilistic methods. Empirical analyses, sustained by statistical methods, can, however, provide more precise indications with higher relevance for practical applications. In this context, a rising level of interest has been addressed to rigorous methods for experimental design of computational tests and statistical analysis of their results. Statistical methodologies are used also for the tuning of the parameters required by stochastic local search algorithms. Unfortunately, besides several papers promoting the use of systematic methodologies in the analysis of empirical data, in practice, papers in combinatorial optimisation which use these instructions are rare.

3.3. Application scenarios

Experiments on algorithms are conducted to gain insights of practical relevance. They are therefore defined in relation to the context of an application. We make a first distinction between two basic contexts: the research context and the practical context.

Research context: Experiments in this context are designed to make powerful statistical inference on the performance of existing algorithms on a well known and possibly standard enough problem. Results should be reusable and should be precise enough for the cases studied. New algorithms must be fairly tested against existing ones. Experiments are also designed to discover or to confirm conjectures about why an algorithm works or why not. In this case, explanations should be made general enough to go beyond the specific problem. There is no pre-defined class of instances to study. Typically, random instances or instances with some structure are deemed good representatives of possible real life applications. In fact, instances as much diversified as possible should be considered, possibly grouped in well defined classes.

Practical context: This is the case of real-life applications where the environment and the instances are usually quite specific. Algorithms must be designed and transformed

into efficient and useful implementations. Often they must be hybridised, configured and tuned. All this process is referred to as *SLS algorithm engineering* and case studies show that experimental techniques are helpful tools to achieve high quality final results.

A second possible distinction concerns the kind of applications for which the algorithm is designed. This distinction determines how fast an algorithm must be in returning a solution. Its understanding is important for the definition of the termination criterion for SLS algorithms, in terms of computation time.

1. In a *design scenario* problems must be solved infrequently and one answer is sufficient to cover long periods. Examples of applications are some typologies of timetabling, like railways or airplane timetables, frequency assignment in the design of network infrastructure, or facility location. For many such problems it is very unlikely that an exact algorithm exists. Quality is critical but computation times of several hours or days are affordable.

2. In a *control scenario* problems must be solved frequently and decisions have a short horizon. Examples are memory allocation, multiprocessor scheduling, routing of data in networks. Exact algorithms usually exist but heuristics are used because answers must be obtained almost in real time. Solution quality is often less important.

3. A *planning scenario* is an intermediate case between the previous two. Some examples are short term timetabling, like weekly employee shift assignments, or production planning. In these cases a solution must be given in short time (from few minutes to few hours) and quality matters. Exact algorithms may exist but are often not appropriate because they are too slow.

In this chapter we present four systematic experimental methods, which can be adopted in all these scenarios. *Design of experiments* with statistical analysis of results is the correct procedure for a research context, in which the analysis of algorithms goes beyond the mere understanding of which is the best. Two further methods, *time dependent profile* and *search landscape analysis*, go even deeper in the analysis of algorithms' behaviour and try to derive some general patterns. *Sequential testing* is, instead, the choice for practical contexts where algorithms must be engineered, and determining the best in the most efficient way is the main issue.

3.4. Performance measurement

The performance of optimisation algorithms that return approximations of optimal solutions, are described by two variables: the quality of the solution and the computation time to produce it.

In the case of SLS methods, a natural termination criterion does not exist, and the longer time is allocated the better the solution quality should be. In order to carry out the comparison among several algorithms, it is then common praxis to allow all algorithms to consume the same amount of computational resources (Rardin and Uzsoy, 2001 refer to this as the "fairness principle") and to restrict the attention to the solution quality. (In

such an experimental setting, algorithms with a natural termination condition should be restarted if they end before the time limit imposed.) If the comparison is done on more than one instance, an instance independent measure of solution quality must be defined.

A convenient measure for *solution quality* is the "distance", or *error*, from the optimal value. This measure exhibits two problematic issues. The first is the determination of the optimal solution, given that, for the cases of our interest, exact methods are infeasible. There may be cases in which the optimal solution is known from the process of construction of the instances. An alternative possibility is the substitution of the optimal solution quality with bounds or approximations. Unfortunately, these values are often weak indicators of the optimal values. *Statistical estimation techniques* based on extreme value theory constitute another alternative for the estimation of optimal solutions. Accordingly, the distribution of the best solutions in n independent solution samples is approximated by a Weibull distribution and a confidence interval for the optimal solution quality is derived from there. These techniques have been applied in the field of combinatorial optimisation (McRoberts, 1971; Dannenbring, 1977; Golden and Alt, 1979; Smith and Sucur, 1996; Ovacik et al., 2000), but further investigations on more problems are necessary to assess the actual reliability of their estimates. A last possibility, widely used in common practice, is the comparison of results with best known solutions. To this end, instances are used that belong to well known benchmark sets for which a collection of good solutions is made available by previous studies. If instead instances are new, then as best solution can be taken the best produced by any of the algorithms involved in the comparison. It might also be a choice to perform long time runs of a good algorithm and record the best solutions found, unless this is computationally too expensive. The drawback of relating the analysis to best known results is that the comparison becomes biased by these values and it may change if the best known values improve.

The second problematic issue in the definition of a measure for the error is that different instances exhibit different scales of solution costs. If we denote the cost of the solution found on a run of an algorithm on an instance i as $c(i)$ and the optimal cost, or a possible approximation of it, as $c_{opt}(i)$, the relative error $|c(i) - c_{opt}(i)|/c_{opt}(i)$ may help to make results among instances more comparable. Zemel (1981) defines an error measure as "proper" if it remains invariant under some trivial transformation on the instance that leaves the problem equivalent. The relative error is not always proper and he proposes as a more robust measure

$$e(c, i) = \frac{c(i) - c_{opt}(i)}{c'(i) - c_{opt}(i)} \tag{3.1}$$

where $c'(i)$ is the worst solution cost. Unfortunately, deriving the worst solution cost may be a problem as hard as finding c_{opt}. Zlochin and Dorigo (2002) suggest then the use of a surrogate value for $c'(i)$, that is, the expected cost of a uniform distribution of solutions or the expected cost produced by a most standard algorithm, such as a heuristic or a random solution generator. With this latter choice, besides being more invariant than the relative error, the error measure 3.1 has also the practical property of providing an immediate indication of how much better an algorithm performs compared to an elementary algorithm. A value of the error $e(c, i)$ close to 1 indicates that the performance of the two algorithms are similar.

A different approach from considering an error measure is to transform results into *ranks*. This method is appealing in experiments with algorithms on several instances because each instance can be seen as a judge who assigns a vote to the algorithms. A ranking procedure operates within instances and assigns value 1 to the best result, value 2 to the

next, etc., with duplicate ranks allowed in the case of identical results. Chiefly important, ranking on instances provides results which are invariant with respect to different instance scales. Nevertheless, ranking necessarily reduces the information available, as it neglects the entity of differences among algorithms on single instances.

If optimisation algorithms are evaluated on the basis of their ability to solve an instance to optimality or to produce a given solution quality, performance can be assessed by measuring the *computation time* (also called *run time*). In order to make run times on different machines comparable, transformation ratios may be used which are obtained by running on all the machines a benchmark code that implements similar algorithmic operations as those of the algorithms studied. When possible, measuring basic algorithmic operations which are common to all algorithms, removes machine dependencies and is, therefore, a better choice.

3.5. Statistical analysis

Statistics is the field of mathematics that studies the probability of events on the basis of inference from empirical data. We distinguish between descriptive statistics and inferential statistics.

In *descriptive statistics*, data are collected and used for descriptive purposes only. Procedures for presenting and summarising data are, for example, tables, graphs and measures of central tendency and variability. Common measures of central tendency are data's *mean* and *median*. Common measures of variability are the *range*, the *quantiles*, the *inter-quantile ranges*, the *standard deviation*, and the *variance*. *Box-plots* permit, instead, to visualise contemporaneously several elements of the whole distribution of data. We use a box to represent the data between the two quartiles in the distribution, and a cross to represent the median value. "Whiskers" are then extended to data points that are no more than 1.5 times the interquartile range away from the boxes while extreme data outside these boundaries are represented by circles. An example is given in Figure 3.1.

In *inferential statistics*, data are employed to make inference or predictions about one or more populations from which the samples have been drawn. Whereas a *population* consists of the total subjects that share something in common, a *sample* is a set of subjects which have been drawn from that population. To differentiate characteristics of the sample, called *statistics*, from characteristics of the population, called *parameters*, it is common practice to use Latin letters and lower Greek letters. Thus, for example, μ and σ^2 are respectively the mean and the variance of a population while $\bar{X}$ and s^2 are respectively the arithmetic mean and variance of the sample. To be useful for making inference, the sample must be representative of the population. A *random sample* is usually satisfactory. Once the sample is taken, the two main methodologies used in inferential statistics are *hypothesis testing* and *parameter estimation*.

In *Hypothesis testing* two statistical hypotheses are evaluated: the *null hypothesis*, commonly represented by the notation H_0, and the *alternative hypothesis*, represented by H_1. The null hypothesis is the statement of no difference between population parameters that the researcher expects to reject, while the alternative hypothesis states the presence of a difference. According to the type of alternative hypothesis, the test can be one-sided or two-sided. The testing procedure consists in evaluating an appropriate *test statistic* S and determining if the value of S from the observed sample is highly unlikely to occur under the null hypothesis. Within this procedure, it is possible to commit two types of errors.

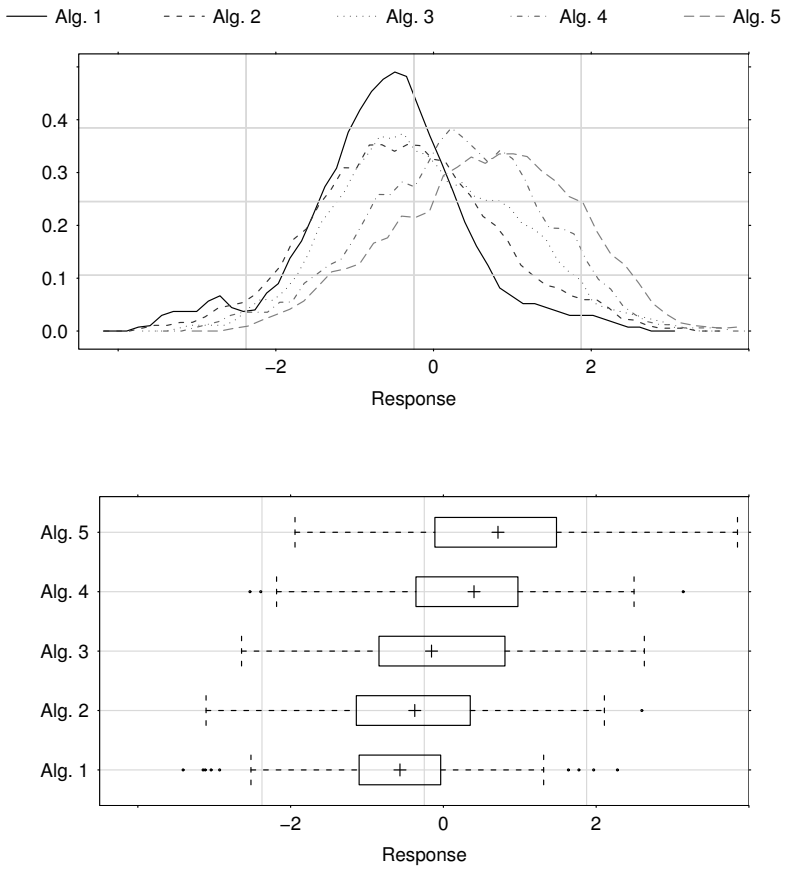


Figure 3.1.: Empirical distributions and box-plots for the results of 5 algorithms on 5 instances. The 30 results for each algorithm–instance pair are sampled from standardised normal distributions under the alternative hypothesis of algorithms with different performance. The reader should compare this representation with the one after the statistical analysis of Figure 3.3.

An error of Type I occurs when a null hypothesis is rejected, although it is true. The likelihood of committing this type of error is specified by the α level assumed *a priori* in the evaluation of the experiment (α determines the level of significance in the statistical inference). An error of Type II occurs when a false null hypothesis is not rejected. The likelihood of committing this type of error is denoted by β. The likelihood of rejecting a false null hypothesis, $1 - \beta$, represents the *power* of the statistical test. The power of a test for a fixed α depends on the statistic, the sample size, and the alternative hypothesis.

In *parameter estimation*, statistical inference is carried out by determining a range of values within which the true value of a population parameter falls with a given degree of confidence. Such a range of values is referred to as a *confidence interval*. As an example, a 95% confidence interval for a population mean indicates that there is 95% of probability that the true mean of the population falls in that interval.

We, finally, remark that the separate use of descriptive and inferential statistics may lead to weak conclusions. If samples are large, inferential statistics are likely to detect even very small differences between populations but these small differences may have no practical relevance. On the other hand, practically relevant differences observed through descriptive statistics should not be claimed without guarantees on their statistical significance. A correct analysis of experimental results must therefore necessarily combine

both, descriptive and inferential statistics.

3.6. Design and analysis of experiments

A basic computational experiment to compare algorithms is collecting several runs for each algorithm on a single instance. More reasonably, algorithms may be tested on several instances and the experiment can be visualised as a matrix with algorithms in the columns and instances in the rows. Instances may be completely independent or grouped by structural characteristics such as size.

In order to make experiments effective, a precise methodology has been developed in a branch of statistics known as *experimental design* (Dean and Voss, 1999). We review the main steps of this methodology with reference to our specific context.

3.6.1. Experimental design

Statement of the objectives of the experiment
A list of the questions to be answered by the experiment must be made explicit. The interest may be merely on the comparison of different algorithms or it may be extended to investigate how instance characteristics affect the algorithms. Another possible interest may be the analysis of algorithms components and their effect on algorithm performance.

Identification of sources of variance
Source of variations are all the components that cause observations to have different numerical values. It is possible to distinguish two type of sources: those that are of particular interest to the experimenter, called *treatment factors*, and those that are of no interest, called *nuisance factors*.

Treatment factors are high-level components whose influence on the data is to be studied. They can be defined as *numerical* or *cathegorical* variables, this latter if there are two or more categories, but there is no intrinsic ordering to the categories. The *levels* of a treatment factor are the specific values or cathegories actually used for the treatment factor in the experiment. For instance, the tabu length in a Tabu Search algorithm is known to have an influence on its performance; hence, it may be considered as a treatment factor, while the specific numerical values assigned to it define the levels of the treatment. Besides tabu length, also the aspiration criterion has an influence; then we may want to consider an experiment with two treatment factors, namely the tabu length and the aspiration criterion. The levels of the aspiration criterion are *categorical*, *i.e.*, "present" and "not-present". The combinations of levels are usually called *treatment combinations* and an experiment involving two or more treatment factors is called a *factorial experiment*. In a *single factor experiment* the term *treatment* is commonly used to denote a level of the treatment factor.

Nuisance factors are factors not explicitly controlled in the experiment. In the context of optimisation, the instances of the problem to be solved constitute a nuisance factor as, for example, in the general case no *a priori* knowledge is given on their optimal solution or on their hardness. For each treatment combination, an instance must be chosen for

testing, thus defining thoroughly the *experimental unit*. Usually, units of a nuisance factor are distributed at random (*randomisation principle*). For example, each algorithm could receive a different, randomly chosen, instance. A different approach, called *blocking*, is however more appropriate in the context of comparisons of algorithms. Each single instance is identified as a *block* and the response of each algorithm is observed the same number of times on each block. This gives rise to a *complete block design*.[1] Rardin and Uzsoy (2001) point out that well designed computational experiments for algorithms always block completely on instances.

It may be possible that we identify more than one nuisance factor, as there are two or more sources of variations that had been supposed to have an effect but that may not be controlled. In this case nuisance factors become *stratification variables* and blocks arise from their combinations. In the analysis of algorithms, two typical stratification variables are size and structure of the instances. The instances remain the blocks of a single blocking factor but they can be "stratified" by the size and structure variables.[2]

With stochastic algorithms, successive trials can produce quite different outcomes. To avoid being mislead by results which are very different from expected, it is common practice to collect *replicates*[3] of the observations in each experimental unit, that is, several *runs* are performed with different random seeds. As we discuss next, replicates may not be necessary if it is possible to have many instances. McGeoch (1992) points out that, in order to reduce the variance of results, a common random seed among algorithms should be used for each replicate.

Definition of the test instances

Rardin and Uzsoy (2001) distinguish four kinds of instances: real world instances, random variants of real instances, online libraries and randomly generated instances. Real world instances are particularly appropriate for real applications but usually only few data are available and the test for the efficacy of the algorithm proposed remains re-

[1] More generally, in experimental design the objects to which treatments are applied in the experimental unit are referred to as *subjects*. In a *between-subject* design, different subjects serve in each of the experimental conditions while in a *within-subject* design, each subject serves under all the experimental conditions. A within-subject design corresponds to a complete blocking design. A design involving *matched-subjects* is also treated as a within-subjects design. In a matched-subjects design, each subject is paired with one or more other subjects who are similar with respect to one or more characteristics that are highly correlated with the response variable. In this sense, an instance solved under all algorithmic conditions may be considered a matched-subject. A complete block design is only possible when the number of homogeneous subjects, which constitute the single block, v, are multiples of the number of treatments, k. Historically, the case with $k = v$ is called *randomised complete block design*, while the cases with larger v are called *general complete random design*. We do not need these distinctions here, since the same instance can be reused on all treatments within the single block.

[2] In the more general context of experimental design, two nuisance factors may be *crossed*, if the experiments to be carried out are too many and running a complete design is too costly. A design with crossed nuisance factors is represented by a Latin square. In a Latin square, the rows and the columns of the matrix correspond to the two nuisance factors, and each cell, which corresponds to a block, receives a treatment level. The number of resulting experiments is considerably reduced, although some good characteristics are maintained. Indeed, the particularity of Latin square design is that, if column headings are ignored (if a blocking factor is removed), the design looks like a randomised complete block design; similarly, if the row headings are ignored, the design with columns as blocks looks like a randomised complete block design. Note that generating Latin squares implies solving a graph colouring problem, as we will see in Chapter 4. In a Latin square design, all treatment and nuisance factors, must have the same number of levels. A possible generalisation are Youden Designs (Dean and Voss, 1999).

[3] In the language of experimental design, the number of replicates is given by the number of times an original observation is replicated. Hence, it does not include the original observation itself. In this thesis however, we assume that the number of replicates corresponds to the number of runs without distinguishing between original observation and replicate.

stricted to the particular context with small general relevance. Random variants of real instances try to maintain the main structure of the instance varying the details. This possibility received however little attention in practice. The vast majority of the literature measures the efficiency of algorithms on benchmark instances from online libraries. Such libraries are an invaluable tool since they allow immediate comparisons with other published works. However, they may have the following pitfalls: (i) they might not be representative of any real application, (ii) they might have been designed for illustrating pathological behaviours, (iii) they might have been chosen because they are particularly suitable for some algorithms, and, (iv) they may induce an over-tuning of algorithms on those instances, reducing the effort to study the behaviour of the algorithms under various situations. Completely random instances are the last alternative when not enough data for the problem under study are available. Random instances have several advantages: (i) they can be generated in large number, (ii) their characteristics may vary, thus widening the study of algorithm behaviour, (iii) if generators are well documented the features of the instances may be known thus allowing to discover relationships between algorithms and problems, and (iv) in some cases, optimal solutions or lower bounds may be known from the construction process. On the other side, they are not representative of any real application and they may induce an over-tuning of algorithms over the particular generation process.

Birattari (2004b) investigates the problem of testing stochastic algorithms on problem instances from a machine learning perspective. He points out that the practical relevance of testing algorithms is to forecast the performance on new future instances. However, he observes that in real applications instances are not all equally likely to appear. He then introduces a formal definition for the concept of classes of instances. This definition is based on a probabilistic model, in which each instance is identified by its probability of appearing. Following this model, the performance of an algorithm over a class of instances is described by a stochastic variable determined by the probability distribution of results on a given instance and the probability distribution of the instances in the class. The expected performance of an algorithm on a class of instances is then the average performance attained on each single instance weighted by the probability that an instance has to occur.

Two important consequences arise from this probabilistic model. First, algorithms should be selected and configured on classes of instances which are representative samples of the distribution of instances that the algorithm will be called to solve in practice. The identification of this representative sample of instances is an important aspect for a realistic assessment of algorithm performance and must be taken into account when "engineering" the algorithm. Secondly, the best experimental setting for estimating the expected performance of a given algorithm, on the basis of a given number of experiments N, can be derived analytically. Contrary to a popular belief, there is no trade off between the number of runs and the number of instances. The setting "one single run on N different instances" guarantees that the variance of the estimate is minimised. Any other experimental setting fails being equally efficient in terms of the reduction of the variance. This result is proven in Birattari (2004a).

Selection of the combinations of factor levels to test

Depending on the computational power available and the duration of the experiment it must be decided between the two alternatives: *full factorial experiments* and *fractional factorial experiments*. The former alternative consists on running all combinations of algorithm factors on all the instances. Usually, computer experiments are not too costly in terms of

time and this design is more likely to detect statistical differences. The latter alternative is mainly used in engineering (see Montgomery, 2000). It consists in choosing a subset of factor level combinations such that effects are not confounded. We will not consider this design in this thesis.

Refinement of the experimental design

Running a pilot experiment is a good practice which may help to better define the experiment. A pilot experiment also helps to identify ceiling or floor effects. Ceiling effects arise when test instances are insufficiently challenging, while floor effects arise in the opposite case of instances which are equally hard to solve. In these cases, it is very hard to gather any statistically significant conclusion and those instances may be removed. It may also become clear that some levels of factors do not have any impact on the observations or that some values assigned to the levels must be better rescaled. In experiments involving algorithms, pilot experiments are also useful to make sure that no bug is present and that all algorithms work under the same conditions. For example, detecting a correlation between the number of runs and the performance of the algorithm may indicate that memory is not correctly deallocated between repeated runs. Finally, from pilot experiments it is possible to estimate the number of replicates that are necessary to attain a desired level of statistical power.

Outline of the analysis

From the definition of the factors involved in the experiment it is possible to hypothesise a model to put in relation the response variable with the sources of variation. Commonly a linear relation is assumed. The simplest model arises in a single factor design and is expressed in the form

$$\text{Response} = \text{constant} + \text{effect of treatment} + \text{error}.$$

A complete block design is instead expressed in the form

$$\text{Response} = \text{constant} + \text{effect of block} + \text{effect of treatment} + \text{error}.$$

The models are used for testing the statistical hypotheses. The outcome of this kind of analysis is an indication whether the treatments or other factors have statistically significant influence on the response variable. If the interest is at a finer level of analysis, meant to establish differences among specific treatments, then a different procedure for multiple comparisons is undertaken.

The kind of analysis depends on the assumptions concerning the populations under analysis. If there are reasonable elements to assume a certain probability distribution, then a parametric analysis may be appropriate. In contrast, if those assumptions are uncertain, a non-parametric analysis is safer. This choice may have an impact also on the number of replicates to collect in the experiment. For example, collecting many replicates (more than 30) makes a parametric analysis more practicable.

Once the experiment has been designed and the data has been collected, results must be analysed. In the following sections we define the statistical methods for the correct analysis.

3.6.2. Statistical tests

We distinguish three types of statistical tests to draw inference on a pair of statistical hypothesis H_0 and H_1: *parametric*, *permutation*, and *rank-based* tests.

There is a general agreement among statisticians that data should be evaluated with the appropriate *parametric test*, if there is no reason to believe that some assumptions, on which these tests are based, are violated (Sheskin, 2000). A procedure for parametric test is outlined in Algorithm 3.1. Parametric tests are the most powerful tests when observations are independent and identically distributed with a known theoretical distribution.

1. Choose a test statistic, F, whose probability distribution under H_0 $p(F)$ is known independently from the observations;
2. Compute F relative to the observations X;
3. Compare $F(X)$ with the upper α-percentage point of $p(F)$ (in case of one-sided tests) and accept or reject H_0 according to whether $F(X)$ is smaller or larger than this value.

Algorithm 3.1: Typical procedure of parametric tests.

When the probability distributions of the populations involved in the comparison are unknown, or heavy tailed, or asymmetric, it is possible to derive the distribution of a test statistic from the empirical observations by using *permutation tests* (Good, 2000; Pesarin, 2001). Generating all possible distinct permutations of data, it is possible to derive the probability distribution of a chosen test statistic and check whether the test statistic of the original observations yields a value which is highly unlikely to appear under the null hypothesis. The test achieved may thus be *exact*, that is, the type I error of the test is exactly the *a priori* chosen significance level. This test procedure is outlined in Algorithm 3.2.

1. Choose a test statistic T;
2. Compute the observed value of the test statistic $T : T_0 = T(X)$ from the original set of observations;
3. Obtain the permutation distribution of T under H_0 by generating all possible rearrangements (corresponding to the permutations) of the observed data and compute the value of the test statistic T on each rearrangement X^*: $T^* = T(X^*)$. (Rearrangements correspond to exchanging a number of observations between the treatments that are compared. With two or more samples, all the observations are combined into a single large sample before being rearranged);
4. Obtain the upper α-percentage point z (in case of one-sided tests) of the permutation distribution of T and reject H_0 if T_0 for the original observations is smaller or larger than the z value.

Algorithm 3.2: Typical procedure of permutation tests.

If the sample size is not very small, generating all possible permutations at the third step becomes computationally prohibitive; in this case, an approximate test can be obtained by a sampling without replacement from the space of all possible permutations.

With a sample of size B, the value z in the fourth step is given by $\#\{T^* \geq z\}/B = \alpha$ (for one-sided tests). Approximate permutation tests are also known as *randomisation methods* (Cohen, 1995). More precisely, (Pesarin, 2001) defines them as *Conditional Monte Carlo methods* where the term *conditional* emphasises that the re-sampling procedure is conditional to the observed data. In the implementation of these methods, each distinct permutation must have the same probability to occur and this requires careful attention in the implementation. We address some of these implementation issues in Appendix B. Various sources (Good, 2000; Pesarin, 2001; Anderson and ter Braak, 2003) suggest using at least samples of size 1000 for tests with an α level of 0.05.

Rank-based tests are permutation tests applied to the ranks of the observations rather than their original values. Since for a given sample size, ranks have the same values independent of the actual values of the observations, the significance levels of the test statistics can be tabulated and heavy computations avoided (Hollander and Wolfe, 1999). Apparently, rank-based tests are less powerful than permutation tests and parametric tests but the same power may be reached by increasing the number of observations (see Good, 2000 and the Appendix for more details on this issue). The loss of power is due to the loss of information caused by the transformation of data. Nevertheless, rank-based tests remove problems related with the normalisation of results and with the presence of outliers in the samples which may affect the exchangeability of observations. The procedure for a rank-based test is outlined in Algorithm 3.3.

1. Choose a test statistic S;
2. Replace the original observations $X_{ij}, i = 1, \ldots, k, t = 1, \ldots, n$ by their ranks in the combined sample $R_{ij} = R(X_{ij}), k = 1, \ldots, kn$ and compute the observed value of the test statistic $S : S_0 = S(R_{ij})$;
3. Compare S_0 with the upper α-percentage point (in case of one-sided tests) of the tabulated distribution and reject H_0 if S_0 is larger than this value.

Algorithm 3.3: Typical procedure of rank-based tests.

Common statistical packages contain pre-implemented procedures to compute the parametric tests as well as rank-based tests. Rare are, instead, packages with methods for permutation tests, one of the reasons being that the test statistic can vary arbitrarily and that these procedures have not yet been tested enough. The GNU project R, a language and environment for statistical computing and graphics, makes possible the implementation of such tests both in its dedicated language or in an embedded language such as C to which it may interface. The second choice has to be preferred, as an optimisation of the code is an important issue for the heavy computations required in some cases by permutation tests.

3.6.3. All-pairwise comparisons

It is usually the case, when comparing algorithms, that the interest is on making inference on multiple hypotheses rather than a single general hypothesis. In this case, the interest is on the statistical significance of each single pair-wise comparison. In an *all-pairwise comparison* each treatment is compared with every other treatment and a family of null hypotheses is tested. The null hypotheses, for example, concern the differences of pairs

of parameters μ_i and μ_j with $i \neq j$, that is, $H_{0ij} : \{\mu_i = \mu_j\}$, for all $i < j$, $i = 1, \ldots, k$, $j = 1, \ldots, k$. This entails testing $c = k(k-1)/2$ hypotheses.

In the context of multiple comparisons the probability of rejecting at least one null hypothesis when it is true is called *family-wise error rate* (Hochberg and Tamhane, 1987).[4] The family-wise error rate α_{FW} is equal to $1 - (1 - \alpha_{PC})^t$ where α_{PC} is the error rate at each single test and t the total number of tests (or hypotheses tested). In order to control the *family-wise error rate* at the desired level of confidence (for example, 0.05) the level of confidence per comparison must be adjusted. When comparisons are planned before collecting the data, the control of the family-wise error rate requires a lower degree of adjustment. This is the case of some specific types of comparisons called *multiple comparison with the best* or *multiple comparison with a control* or the more general *multiple comparison of contrasts*, *i.e.*, linear combinations of means (we refer the reader to Dean and Voss (1999) and Hsu (1996) for a full treatment of these cases). An all-pairwise comparison, instead, maximises the number of comparisons conducted and requires a stronger adjustment, as t becomes equal to $c = k(k-1)/2$ (Hsu, 1996). Yet, there is a lack of consensus on the degree of adjustment to undertake because of the possible presence of implications (and hence dependency) between some of the hypothesis. Thus, a variety of methods for adjusting α_{PC} have been proposed.

The basic method treats all comparisons as independent and consequently adjusts the per-comparison error rate as $\alpha_{PC} = 1 - (1 - \alpha_{FW})^{1/c}$. Such adjustment is quite conservative.[5] An even more conservative approach is the *Bonferroni's method*. Based on the fact that $(1-\alpha)^{1/c} < 1 - \alpha/c$, Bonferroni suggests to adjust $\alpha_{PC} = \alpha_{FW}/c$. The difference between $(1-\alpha)^{1/c}$ and $1 - \alpha/c$ is however typically very small and no practical difference exists between the two methods. Slightly more complex are step-wise approaches like the sequentially rejective *Holm's method*. In this method the p-values obtained by the c tests are ordered in nondecreasing order and compared in order against the levels $\alpha_{FW}/c, \alpha_{FW}/(c-1), \ldots \alpha_{FW}/1$ (Holm, 1979). Both, Bonferroni and Holm's methods give a strong control of the family wise error rate and in general the Bonferroni's correction is dominated by Holm's method, which is also valid under arbitrary assumptions. Other possible methods are step up methods and closed testing methods. For more details we refer to Hochberg and Tamhane (1987), Shaffer (1995), Hsu (1996), and Ekenstierna (2004). Here, we will mainly focus on methods that yield an equal α_{PC} on all the hypotheses tested. The reasons for this choice are related with the graphical representation of results that we intend to produce.

Multiple comparisons can be carried out by means of *simultaneous confidence intervals* or *simultaneous tests* (Hsu, 1996). Computing simultaneous confidence intervals offers the possibility to represent results graphically and to infer the significance of differences by visual inspection. In this respect, we endorse the view of the statistician J. Tukey: "the best single device for suggesting, and at times answering, questions beyond those originally posed is the graphical display".[6] We will, therefore, give preference to multiple comparisons procedures for simultaneous confidence intervals and only mention, when necessary procedures for multiple testing.

[4] More precisely a family-wise error rate in *strong sense* defines the probability of rejecting at least one null hypothesis when it is true while a family-wise error rate in *weak sense* makes reference to the probability of rejecting the global null hypothesis only (*i.e.*, $\mu_1 = \ldots = \mu_k$). We avoid to consider the *weak sense* and refer only to the *strong sense*.

[5] The term conservative is related to the power of the test. Decreasing α_{PC} the power $1 - \beta$ also decrease making smaller the probability of detecting significant difference when there is actually one.

[6] D. R. Brillinger, "John Wilder Tukey (1915-2000)", Notices American Mathematical Society. 49 (2) (2002), 193-201.

Algorithm 1	Algorithm 2	...	Algorithm k
X_{11}	X_{21}		X_{k1}
X_{12}	X_{22}		X_{k2}
$\vdots$	$\vdots$		$\vdots$
X_{1r}	X_{2r}		X_{kr}

Figure 3.2.: Design with several algorithms and several runs per algorithm on one single instance.[8]

We proceed by reviewing the statistical tests for three different experimental designs: *several runs on one single instance*, *one single run on various instances*, and *several runs on various instances*. The first is not relevant for us but it serves as an introductory case.

3.6.4. Design A: Several runs on one single instance

In this first design, we compare k algorithms on the basis of r *runs* collected on one single instance. Each algorithm constitutes one of the k treatments of the factor. The data consist of k random samples of equal size r, one sample per algorithm. We denote the i-th random sample by $X_{i1}, X_{i2}, \ldots, X_{ir}$ and X_{it} the replicate t of algorithm i. Then, the data can be represented in the columns of a matrix as shown in Figure 3.2.

Each measured response X_{it} may be expressed as the sum $\mu_i + \epsilon_{it}$, where μ_i is the "true response" and ϵ_{il} an error given by the presence of nuisance variation. If we assume that the variance of the response is due to the differences in means of the responses of the algorithms, the model can be rewritten as $X_{it} = \mu + \alpha_i + \epsilon_{it}$, where α_i are the *algorithm effects*, corresponding to the deviations from μ due to the i-th algorithm. Note that the only assumption of this model is the linearity of the relation.

The question of interest is whether the algorithms differ in terms of their effects on the response variable. This corresponds to test the general null hypothesis $H_0 : \{\alpha_1 = \alpha_2 = \ldots = \alpha_k\}$ and the alternative hypothesis is $H_1 : \{\text{at least one of the } \alpha_i \text{ differs}\}$.

More specifically, the interest may be on the all-pairwise comparisons. In this case the family of null hypotheses is given by $H_{0(ij)} : \{\alpha_i = \alpha_j\}$ for $i < j, i = 1, \ldots, k, j = 1, \ldots, k$ and of the alternative hypotheses by $H_{1(ij)} : \{\alpha_i \neq \alpha_j\}$ for $i < j, i = 1, \ldots, k, j = 1, \ldots, k$.

Parametric tests

The analysis studies the reasons of variance in the linear model above and it is commonly referred to as ANOVA (Dean and Voss, 1999; Montgomery, 2000). Let $\bar{X}_{i.} = \sum_{t=1}^{r} X_{it}/r$ and $\bar{X}_{..} = \sum_{i=1}^{k} \sum_{t=1}^{r} X_{it}/kr$. The sum of squares of the deviations around the grand mean $\bar{X}_{..}$ may then be decomposed into two sums

$$\sum_{i=1}^{k} \sum_{t=1}^{r} (X_{it} - \bar{X}_{..})^2 = \sum_{i=1}^{k} \sum_{t=1}^{r} (X_{it} - \bar{X}_{i.})^2 + \sum_{i=1}^{k} r(\bar{X}_{i.} - \bar{X}_{..})^2$$

[8] This design has the algorithms as single factor and may also be referred to as *several independent samples* (Conover, 1999), *single factor between-subjects* (Sheskin, 2000), *one way analysis of variance* (Dean and Voss, 1999; Montgomery, 2000), and *one way layout* (Hollander and Wolfe, 1999).

the first of which represents the *within-group* sum of squares, and the second the *between-group* sum of squares.

The statistic for testing that the main reason of variance are the k algorithms rather than the random variation in the replication of the observations is given by the F-ratio (also known as Snedecor's F) of the adjusted *between-group* variance on the *within-group* variance. Introducing the abbreviations MSA and MSE for the *mean square per algorithm* and the *mean square per error*, respectively, the F-ratio is given by

$$F = \frac{\text{MSA}}{\text{MSE}} \qquad \text{MSA} = \frac{\sum_{i=1}^{k} r\left(\bar{X}_{i.} - \bar{X}_{..}\right)^2}{k-1} \qquad \text{MSE} = \frac{\sum_{i=1}^{k}\sum_{t=1}^{r}\left(X_{it} - \bar{X}_{i.}\right)^2}{kr-k} \tag{3.2}$$

In general, large enough values for the F-ratio imply that the *between-group* variance is the main reason of variance and, hence, there is a strong effect of the treatments.

Under the assumption that the results of the algorithms are *independent*, *normally distributed* and with *equal variance*, *i.e.*, *homoschedastic*, the numerator and denominator of the F-ratio follow a χ^2 distribution and the F-ratio follows the Fisher distribution F_{df_1,df_2} with $df_1 = k-1$ and $df_2 = r-k$ degrees of freedom.

In order to check whether the assumption of normality of the distributions of the data are reasonable it suffices to examine the distribution of the *standardised residuals* from the fitted linear model. The residuals are the observed values $\hat{e}_{it} = X_{it} - \hat{X}_{it}$ where $\hat{X}_{it}$ are the estimated values from the least squares regression of the linear model. The standardised residuals reflect the underlying distributions of data and if they are independent, homoschedastic among the algorithms, and normally distributed with mean equal to zero and variance equal to one then the ANOVA test is applicable. The inspection of these properties may be done visually. If normality does not arise, it may still be possible to apply some transformation of the data, the most common are square, square-root, logarithm, and inverse, or to remove outliers from data if these are deemed influential in the analysis.

In parametric all-pairwise comparisons, confidence intervals for the differences of the algorithm sample means are derived. Then, any pair of algorithms i and j with mean response μ_i and μ_j are declared significantly different at a family-wise level α_{FW} if $|\bar{X}_i - \bar{X}_j| > \text{MSD}$, where $\bar{X}_i = \sum_t X_{it}/r$ and MSD is the *minimal significant difference*. Several parametric methods to derive MSDs have been proposed in the literature. We will focus on the *Tukey's Honest Significant Difference (HSD) method* that, according to Hsu (1996) is particularly suitable for all pairwise comparisons. The minimal significant difference between sample means is computed as

$$\text{MSD}_{HSD} = q_{(\alpha_{FW}/2,k,k(r-1))}\sqrt{\frac{\text{MSE}}{r}} \tag{3.3}$$

where q_{α,df_1,df_2} is the Studentized range distribution. The Tukey test is very similar to a *t* test, where the *t* distribution is substituted by the Studentized range distribution that automatically adjusts the comparison-wise error rate α_{PC} by controlling the family-wise error rate. More specifically, the relation to the *t* test with Bonferroni's adjustment for multiple comparisons (also known as *Fisher's Least Significant Difference*[9]) is $q_{\alpha_{FW},df_1,df_2} \approx$

[9]Often, in the literature, the Fisher's least square significant method is also applied without adjustment. In this case, it is said to be protected if it is applied only after the rejection of the general null hypothesis, and unprotected if the general null hypothesis is not checked. Compare with Montgomery (2000) and Hsu (1996).

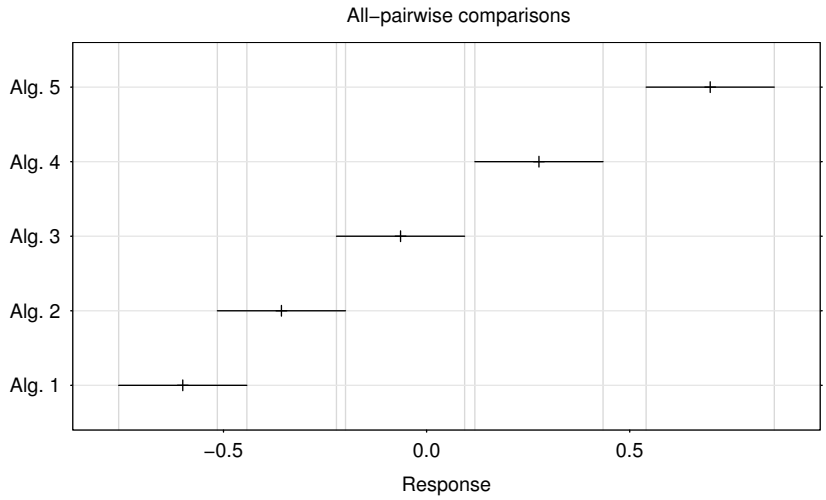


Figure 3.3.: All-pairwise comparisons by means of simultaneous confidence intervals for the average response. The spread of the intervals is computed by means of the Tukey formula; the distributions of the original data is given on Figure 3.1.

$\sqrt{2} \cdot t_{\alpha_{PC},df_2}$. If $k = 2$, the Tukey's method coincides with the *t test*. For a description of other methods like the *Scheffé's method*, also appropriate for all-pairwise comparisons, we refer to Hsu (1996) and to Dean and Voss (1999).

A graphical representation of simultaneous confidence intervals based on MSDs is given in Figure 3.3. The graph is obtained by attaching error bars to a scatter plot of the estimated effects versus treatment labels. The lengths of the error bars are adjusted so that the population means of a pair of treatments can be inferred to be different if their bars do not overlap. From the definition of MSD, the bars correspond to $\bar{X}_i \pm \text{MSD}/2$.[10] We remark that this graphical representation is possible only for tests where the simultaneous confidence intervals have the same width for all individual comparisons. Step-wise methods or comparisons between distributions with different variance are not included. This is because there is no guarantee of finding the half-lengths of the error bars when the MSDs are comparison dependent (Hsu, 1996). A different representation to cope with these cases will be discussed relatively to Figure 3.8.

Permutation tests

With permutation tests we may neglect the assumption that the underlying distributions are normal and restrict ourselves to assume that X_{it} are independent and identically distributed (hence, still homoschedastic but not anymore normally distributed) or *exchangeable* with respect to algorithms.[11] Then, under the null hypothesis, all $(kr)!/(r!)^k$ assignments of r observations to the treatments $1, \ldots, k$ are equally likely. For each permutation of data, denoted as X^*, we can compute the test statistic $T^* = F(X^*)$ using the

[10] Note that the statement that the population mean of an algorithm i falls inside the confidence interval $[\bar{X}_i - \text{MSD}/2, \bar{X}_i + \text{MSD}/2]$ at the level of confidence α_{FW} is improper because the width of the confidence interval MSD is derived for the differences of means and not for the single means.

[11] A set of random variables (observed data) is said *exchangeable* if for any permutation of indexes $\pi(1), \ldots, \pi(kr)$ we have $P(X_1, X_2, \ldots, X_{kr}) = P(X_{\pi(1)}, X_{\pi(2)}, \ldots, X_{\pi(kr)})$. While the assumption of identical and independent distributions implies the exchangeability of data the converse is not necessarily true.

F-ratio of Equation (3.2) as in the parametric case. In fact, given that some terms remain invariant under permutation, the statistic can be simplified to

$$T' = \sum_{i=1}^{k} \frac{\left(\sum_{t=1}^{r} X_{it}\right)^2}{r} \tag{3.4}$$

For generating all the permutations data are first pooled in a single vector and then permutations of this vector are considered. In doing this $(r!)^k$ permutations yield the same statistic value and are superfluous. When using Monte Carlo sampling procedures however, the presence of superfluous permutations does not affect negatively the analysis, as equal permutations are equally weighted for each algorithm and their inclusion does not invalidate the analysis.

To the best of our knowledge, no attempt has been done to derive simultaneous confidence intervals for multiple comparisons from permutation tests. Pesarin (2001) suggests a procedure for obtaining permutation confidence intervals for the difference of means between two treatments. We extend this procedure to the case of all-pairwise comparisons in Algorithm 3.4. The algorithm constructs the interval by extending an initial width until it reaches a value that satisfies, at a confidence level of $1-\alpha_{PC}$, all individual comparisons. The level $1-\alpha_{PC}$ is set equal to $1-\alpha_{FW}$ (in the appendix we compare this choice we the value adjusted from the family-wise level with Bonferroni and verify empirically that the adjustment is not needed). The procedure of Pesarin (2001) is used for checking whether the minimal significant difference between two algorithms i and j guarantees the declared level of confidence. An initial value for MSD can be the value provided by one of the parametric methods, as, for example, from the t test MSD $= t_{\alpha_{PC},kr-k} \cdot \sqrt{2 \cdot \text{MSE}/r}$, and, in case, made slightly smaller. The final value for the MSD returned by Algorithm 3.4 is then used for the graphical representation of the simultaneous confidence intervals in correspondence to each $\bar{X}_{i.}$.

Rank-based tests

The rank-based counterpart is the *Kruskal-Wallis test*. The collection of all observations X_{it} is ranked *jointly*, $R_{it} = R(X_{it})$, $i = 1, \ldots, k, t = 1, \ldots, r$, with rank 1 given to the best response, 2 to the second best, and so on; in case of ties, ranks are averaged. Then the test statistic S_{KW} is computed as

$$S_{KW} = \frac{1}{V^2}\sum_{i=1}^{k}\frac{R_i^2}{r} - \frac{kr(kr+1)^2}{4}; \qquad V^2 = \frac{1}{kr-1}\Big(\sum_{it} R_{it}^2 - kr\frac{(kr+1)^2}{4}\Big) \tag{3.5}$$

where R_i is the sum of the ranks assigned to the i-th algorithm, that is, $R_i = \sum_t R(X_{it})$. The formula is valid for the general case of the presence of ties in the results. Under the null hypothesis, and for a large enough design size, the distribution of the test statistic S_{KW} is approximated by the χ^2 distribution with $k-1$ degrees of freedom (Conover, 1999).

A procedure for simultaneous confidence intervals in all-pairwise comparisons when all kr observations are ranked *jointly* is derived from the Kruskal-Wallis model. In this case, confidence intervals are computed for the differences between means of ranks $\bar{R}_i - \bar{R}_j$ of two algorithms and these differences are claimed statistically significant if $|\bar{R}_i -$

Procedure Compute_all-pairwise_MSD();
Choose an estimated error ϵ related with B;
Start: Choose a positive number MSD;
for all (i, j) of the $k(k-1)/2$ pairwise comparisons **do**
  if Compute_pairwise_MSD(i, j) returns false **then** goto Start;
end
Return MSD.

Procedure Compute_pairwise_MSD(i, j);

1. Subtract $\bar{X}_{i.} - \bar{X}_{j.} +$ MSD from every value of the data group relative to one of the two treatments, say i, obtaining the new vector $X_{it}(\text{MSD}) = X_{it} - \bar{X}_{i.} + \bar{X}_{j.} - \text{MSD}$, $i = 1, 2, ..., r$. This vector is combined with the vector X_{jt} and constitutes the pool of observation to permute.
2. Compute the statistic $T(\text{MSD})$ for the observed response $T(\text{MSD}) : T_0(\text{MSD}) = \bar{X}_{i.}(\text{MSD}) - \bar{X}_{j.}(\text{MSD})$.
3. By rearranging the observations B times obtain the permutation distribution of the statistic $T^*(\text{MSD}) : \bar{X}^*_{i.}(\text{MSD}) - \bar{X}^*_{j.}(\text{MSD})$.
4. Return "true" if the condition $|\#(T^*(\text{MSD}) \leq T_0(\text{MSD}))/B - \alpha_{PC}/2| < \epsilon/2$ is satisfied, else return "false".

Algorithm 3.4: An algorithm for computing simultaneous MSDs in the case of repeated measures. The confidence level $1 - \alpha_{PC}$ is set equal to $1 - \alpha_{FW}$.

$\bar{R}_j| > \text{MSD}_{KW}$ where

$$\text{MSD}_{KW} = t_{1-\alpha_{FW}/2}\sqrt{V^2 \cdot \frac{kr - 1 - S_{KW}}{k(r-1)} \cdot \frac{2}{r}} \tag{3.6}$$

where V and S_{KW} are derived from Equation 3.5. In the formula, the level of confidence is not adjusted, hence, it is suggested to adopt at least the protection of testing first the general hypothesis (given by Equation 3.5) and continuing only if it is rejected (Conover, 1999).[12]

This method assumes that all kr results are ranked together. Hence, each individual comparison between two algorithms, i and j, depends also on the observations of the other $k - 2$ algorithms. It may then be opportune to corroborate the inference with the *Mann-Whitney test* adapted for simultaneous testing with an opportune adjustment of the α-value. In this test, results are ranked only between the two algorithms involved in the comparison and the more powerful Holm's adjustment procedure may be adopted.

3.6.5. Design B: One single run on various instances

In this case, the data consist of b mutually independent k-variate random variables $X = (X_{h1}, X_{h2}, \ldots, X_{hk})$, with $h = \{1, \ldots, b\}$, b the number of instances (*i.e.*, the blocks), and k the number of algorithms (*i.e.*, treatments). The random variable X_{hi} describes the

[12] Other authors give different formulas (Sheskin, 2000, Hollander and Wolfe, 1999), while Hsu (1996) criticises the joint ranking approach and suggests a method in which ranks are computed for pairs of algorithms.

	Algorithm 1	**Algorithm 2**	...	**Algorithm k**
Instance 1	X_{11}	X_{12}		X_{1k}
⋮	⋮	⋮		⋮
Instance b	X_{b1}	X_{b2}		X_{bk}

Figure 3.4.: Design with several algorithms on various instances and one single measurement.[13]

observation relative to the i-th algorithm on instance h. The data can be arranged in a matrix as shown in Figure 3.4.

The linear model must include the effects of the instances (blocks) θ_i, because in a blocking design their influence on the response variable is assumed important. Hence, the model becomes $X_{hi} = \mu + \alpha_i + \theta_h + \epsilon_{hi}$.

Parametric tests

The test for the general hypotheses is again derived from the decomposition of the variance and relies on the F-ratio of Equation 3.2. The terms of the ratio are now expressed by

$$\text{MSA} = \frac{b\sum_{i=1}^{k}(\bar{X}_{.i} - \bar{X}_{..})^2}{k-1}; \qquad \text{MSE} = \frac{\sum_{h=1}^{b}\sum_{i=1}^{k}\left(X_{hi} - \bar{X}_{h.} - \bar{X}_{.i} + \bar{X}_{..}\right)^2}{bk - b - k + 1} \tag{3.7}$$

Under the assumptions of independence, homoschedasticity, and normality, the F-ratio follows a Fisher distribution with $k-1$ and $bk-b-k+1$ degrees of freedom Dean and Voss (1999). The effect of the algorithms α_i is then statistically important if the computed F is larger than the upper α-percentage point of this distribution.

The value of the MSDs for the simultaneous confidence intervals in the all-pairwise comparisons is given by Tukey's HSD method similarly to the single factor case. The formula is

$$\text{MSD}_{HSD} = q_{(\alpha_{FW}/2,k,bk-b-k+1)}\sqrt{\frac{\text{MSE}}{b}} \tag{3.8}$$

where the only changes are on the degrees of freedom and on the term MSE that now is given by Equation 3.7. Connections for this method may be found to the *t* test with matched-pairs.

Permutation tests

Here observations are exchangeable only "within" instances while they are not "between" instances. The total number of possible permutations is $(k!)^b$, that is, $k!$ possible permutations for each of the b blocks. The test statistic may be the F-ratio or the following, which is easier to compute:

$$T = \sum_{i=1}^{k}\left(\sum_{h=1}^{b} X^*_{hi}\right)^2 \tag{3.9}$$

[13] In the literature this design is also referred to as *single factor within-subjects design* Sheskin (2000) or *one way analysis of variance with complete blocking design* Dean and Voss (1999).

For the implementation of the test, a vector of results is maintained for each instance and the elements of each vector permuted in all possible ways. For an optimisation of the code, the procedure can be implemented in such a way that in each new permutation of data only one vector changes in a minimal-change order. Alternatively, each vector is shuffled randomly in Monte Carlo simulations.

Algorithm 3.4 is easily adapted to compute MSDs in all-pairwise comparisons in the current design. The only change is the way permutations are applied in procedure Compute_pairwise_MSD. Specifically, since permutations can only occur within instances and differences between pairs of data are needed, the permutation distribution of the statistic S can be obtained by generating all possible permutations of b "$+/-$" signs. The use of Gray Code to generate these permutations in minimal-change order (Press et al., 1992) may be useful to increase the size of the experiments for which a exact tests may be obtained without the need to use Monte Carlo simulations.

Rank-based tests

The method of ranks for a two-way design is due to Friedman (1937). The consequent *Friedman test* consists in ranking the data in each row of the matrix of Figure 3.4 and then testing whether rank means for the several columns can be supposed to be equal. The test statistic is

$$S_F = \frac{12}{bk^2} \sum_{i=1}^{k} \Big(\sum_{h=1}^{b} R_{hi} \Big)^2 - 3b(k+1) \tag{3.10}$$

where R_{hi} is the rank entered for the i-th algorithm on the h-th instance. If b and k are not too small, S_F tends to be distributed like a χ^2 distribution with $k-1$ degrees of freedom. When there are only three treatments, instead, a more appropriate test is the Quade test (Conover, 1999).

An issue related to the Friedman test is whether the b sets of ranks are strongly correlated one another or if they are unexpectedly independent. A test to check the significance of a certain correlation is obtained by the ratio of the observed variance of the sum of ranks on the columns and their maximal variance and is called *Kendall's coefficient of concordance* (Sheskin, 2000). This test is however not independent from the *Friedman test*. If instances influence in a different "direction" the results of different algorithms, then the *Friedman test* will not be significant even if the algorithms do not perform the same, for, in such a case, the mean ranks of the k algorithms may all have the same expected value, although the k ranks for each instance do not (Friedman, 1937).

For all-pairwise comparisons, we distinguish two approaches. Based on the Friedman test and the same *joint* ranking procedure, the minimal significant difference between rank means of any two algorithms i and j, $|\bar{R}_i - \bar{R}_j|$, according to Conover (1999), may be computed as[14]

$$\text{MSD}_F = t_{1-\alpha_{FW}/2,df} \cdot \sqrt{\frac{2(b\sum_{hi} R_{hi}^2 - \sum_i \bar{R}_i^2)}{b^2(b-1)(k-1)}}; \qquad df = (b-1)(k-1) \tag{3.11}$$

[14] Daniel (1978), Sheskin (2000), and Hollander and Wolfe (1999) indicate a different approximation based on the standardised normal distribution z or on the Studentized range distribution. We were unable to show analytically that the different formulas coincide. In contrast, the values for the MSD provided by Equation 3.11 result always tighter than those attained by the formulas given by the other authors. Moreover, Hsu (1996) observes that methods for all-pairwise comparisons based on Friedman-type joint ranking may exhibit an actual probability of making at least one incorrect assertion larger than α_{FW}.

	Algorithm 1	**Algorithm 2**	...	**Algorithm k**
Instance 1	$X_{111}, \ldots, X_{11r}$	$X_{121}, \ldots, X_{12r}$		$X_{1k1}, \ldots, X_{1kr}$
Instance 2	$X_{211}, \ldots, X_{21r}$	$X_{221}, \ldots, X_{22r}$		$X_{2k1}, \ldots, X_{2kr}$
$\vdots$	$\vdots$	$\vdots$		$\vdots$
Instance b	$X_{b11}, \ldots, X_{b1r}$	$X_{b21}, \ldots, X_{b2r}$		$X_{bk1}, \ldots, X_{bkr}$

Figure 3.5.: Design with several algorithms on various instances and repeated measures.[15]

To guarantee a minimal control over the family-wise error rate, the application of this formula is recommended only if the general null hypothesis has been rejected through the test statistic of Equation 3.10.

The second approach is the *Wilcoxon matched-pairs* simultaneous testing procedure, which might be used to corroborate the inference on all-pairwise comparisons produced by the Friedman test. Whereas with the Friedman method the k observations are ranked together within each block, and, hence, each individual test between two algorithms i and j depends also from the observation value of the other $k-2$ algorithms, in the Wilcoxon test the comparison between each pair of algorithms is carried out on the basis of *signed-ranks*. These are the ranks from the smallest to the largest absolute matched-differences $D_h = |X_{ih} - X_{jh}|$, with any difference $D_h = 0$ being removed. In this case, since we do not look for MSDs of equal width, the more liberal Holm's adjustment can be used for controlling the error produced by multiple tests.

3.6.6. Design C: Several runs on various instances

For each experimental unit algorithm–instance r, $r > 1$, independent *runs* are collected. Again, the instances are blocks and observations in different blocks are assumed to be independent. Data may be viewed as random samples of size r of b mutually independent k-variate random variables, $X = (X_{hi1}, X_{hi2}, \ldots, X_{hir})$, where k is the number of treatments, b the number of blocks, and r the number of observations per experimental unit. The random variable X_{hit} is the t-th realization on block h for algorithm i. Data may be visualised as in Figure 3.5.

Here, two cases may be distinguished. In a first case, we consider as unimportant the interaction between individual instances and algorithms and the model summarising the experiment is $X_{hit} = \mu + \alpha_i + \theta_h + \epsilon_{hit}$. Nevertheless, the question raised in Section 2.4.5 on whether an algorithm performs the same on all instances of a specific problem remained unsolved and it may be interesting to test whether there actually is an interaction. Collecting more than one observation per instance, makes it possible to test for this effect. This gives rise to a second case in which the model is extended to include also an interaction effect $(\alpha\theta)_{hi}$ between algorithms and instances and it becomes $X_{hit} = \mu + \alpha_i + \theta_h + (\alpha\theta)_{hi} + \epsilon_{hit}$.

[15] This design corresponds to a several treatments with blocking factor and repeated measures design. In the literature it is also referred to as *single factor mixed design* (Sheskin, 2000), and *one way analysis of variance with complete blocking and repeated measures* (Dean and Voss, 1999).

Parametric tests

The test is again derived from the analysis of variance.

In the first case, the terms in the F-ratio are computed as

$$\text{MSA} = \frac{br \sum_{i=1}^{k} (\bar{X}_{.i.} - \bar{X}_{...})^2}{k-1}; \quad \text{MSE} = \frac{\sum_{h=1}^{b} \sum_{i=1}^{k} \sum_{t=1}^{r} \left(X_{hit} - \bar{X}_{h..} - \bar{X}_{.i.} + \bar{X}_{...}\right)^2}{bkr - b - k + 1} \tag{3.12}$$

Then, under the parametric assumptions, the F-ratio follows a Fisher distribution with $k-1$ and $(b-1)(k-1)$ degrees of freedom.

In the second case, besides the effect of the algorithms α_i, we may want to test also the effect of the interaction $(\alpha\theta)_{hi}$. We distinguish the F-ratios needed to do so with F^A and $F^{A\theta}$, respectively. The terms in the numerator and denominator then are

$$\text{MSA} = \frac{br \sum_{i=1}^{k} (\bar{X}_{.i.} - \bar{X}_{...})^2}{k-1} \qquad \text{MSA}\theta = \frac{r \sum_{h=1}^{b} \sum_{i=1}^{k} (\bar{X}_{hi.} - \bar{X}_{h..} - \bar{X}_{.i.} + \bar{X}_{...})^2}{(b-1)(k-1)}$$
$$\text{MSE} = \frac{\sum_{h=1}^{b} \sum_{i=1}^{k} \sum_{t=1}^{r} \left(X_{hit} - \bar{X}_{hj.}\right)^2}{bk(r-1)} \tag{3.13}$$

and under the parametric assumptions $F^A \sim F_{k-1,bk(r-1)}$ and $F^{A\theta} \sim F_{(b-1)(k-1),bk(r-1)}$. Note that the main effect of the blocks is not tested because not relevant, as we cannot control the instances.

All-pairwise comparisons may be carried out again with Tukey's method. For the model without interaction, the formula for the minimal significant difference between the means $\bar{X}_{.i.}$ is

$$\text{MSD}_{HSD} = q_{(k,bkr-b-k+1,\alpha_{FW})} \sqrt{\frac{\text{MSE}}{br}} \tag{3.14}$$

where the term MSE is given by Equation 3.12.

For the model with interaction, the formula for comparing the means $\bar{X}_{.i.}$ becomes

$$\text{MSD}_{HSD} = q_{(bk,bkr-bk,\alpha_{FW})} \sqrt{\frac{\text{MSE}}{br}} \tag{3.15}$$

with the term MSE given by Equation 3.13. However, if the interaction is significant (we suppose to have tested this effect before proceeding to all-pairwise comparison) the analysis of all-pairwise comparisons could also proceed separately for each instance. On each instance h the minimal significant difference for the average response of two algorithms i and j, $\bar{X}_{hi.} - \bar{X}_{hj.}$ is computed from Equation 3.15 with br replaced by r in the denominator.

Permutation tests

Permutation tests for factorial layouts are a current active direction of research of applied statistics (Loughlin and Noble, 1997; Good, 2000; Pesarin, 2001; Anderson and ter Braak, 2003). Recently, the method of using synchronised permutations introduced by Pesarin (2001) is receiving increasing interest. Nevertheless, the correct implementation

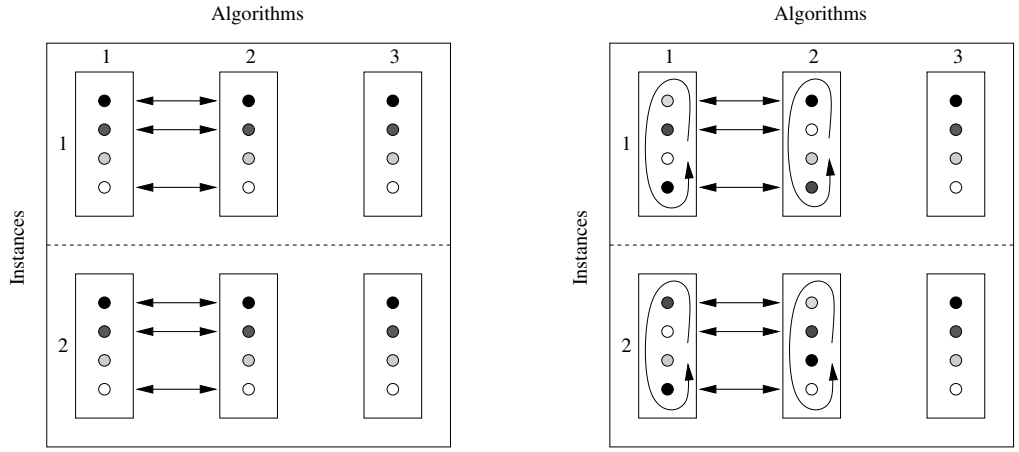


Figure 3.6.: A pictorial representation of the two procedures for generating synchronised permutations: constrained synchronised permutations on the left and unconstrained synchronised permutations on the right. In both cases, 3 observations are exchanged between algorithms 1 and 2, but while on the left the data are maintained fixed in their positions, on the right they are shuffled before the exchange.

of such tests is still an open issue and they must undergo severe simulation analysis before becoming widely accepted. Accordingly, the test for the general hypothesis of an algorithm effect α_i is obtained as follows. An intermediate statistic is computed within each instance h for any two algorithms i and j as $T^*_{ij|h} = \sum_t X_{iht} - \sum_t X_{jht}$. Then, for each instance there are $k(k-1)/2$ intermediate statistics, each comparing two different algorithms. If for each pair i and j the number of swapped observations ν^*_{ij} is maintained equal among all blocks, that is, if permutations are *synchronised* along the instances, then the set of intermediate statistics $\sum_h T^*_{ij|h}$ depends only on treatment effects and on errors (Pesarin, 2001) while the instance and the interaction effects are removed. Thus, the test statistic

$$T^*_A = \sum_{i<j} \Big(\sum_{h=1}^{b} T^*_{ij|h} \Big)^2 \tag{3.16}$$

gives rise to an exact permutation test for the effect of the algorithms, which is valid independently from the presence of interaction. The test statistic for checking the interaction effect is given by

$$T^*_{A\theta} = \sum_{i<j} \Big(\sum_{h<l} T^*_{ij|h} - T^*_{ij|l} \Big)^2 \tag{3.17}$$

which depends only on the interaction effects and on a linear combination of errors. Pesarin (2001) observes that, contrarily to the parametric F test, the two statistics obtained by synchronised permutations are uncorrelated and better satisfy the natural requirement of separating main effects from interactions. This allows to consider permutation solutions competitive even within parametric assumptions.

Pesarin (2001) indicates two possible procedures to generate the synchronised permutations of data: *constrained synchronised permutations* (CSP) and *unconstrained synchronised permutations* (USP). In the first case, elements in exactly the same corresponding coordinates (represented by the index t) are exchanged in all instances. Thus, exactly the same permutation is applied to each instance giving rise to $\binom{2r}{r}$ different restricted permutations from which the values of the intermediated statistic are derived. The same number of values can receive also the statistic T^*_A. In the second case, the same number

of elements are exchanged *after shuffling data in each instance*. This procedure gives rise to $\sum_{\nu^*=1}^{r} \binom{r}{\nu^*}^2$ different restricted permutations for each intermediate statistics. Then, the possible alternative permutations for the statistic T_A^* are $\sum_{\nu^*=1}^{r} \binom{r}{\nu^*}^{2b}$. In Figure 3.6 we give a pictorial representation of the difference between the two procedures. We observe that CSP allow to control the minimal attainable α-size of the test, although in general USP give lower minimal attainable α-sizes. In practice, when the number of design replicates is small (say 2 or 3) and the design includes more than 3 blocks (instances), USP should be preferred. When there are more than 3 replicates, then CSP may be adopted. In both USP and CSP, the permutation distributions of the test statistic in Equation 3.16 are symmetric. In the case of CSP, this fact makes feasible the generation of all permutations until replicates of size 7, as in this case the number of permutations to consider is only 1716. For $r > 7$, instead, the distribution of the statistic may be approximated by Monte Carlo sampling. An exact solution for USP is instead much more complicated and so far only Monte Carlo simulations have been attempted. Nevertheless, in the case of Monte Carlo simulations a problem is to have each different permutation to appear with equal probability. In Algorithm 3.5, we give the algorithm for computing the test statistic on a sampled permutation in both the CSP and the USP scheme.[16] The procedures must then be repeated for a chosen number B of Monte Carlo simulations or, in the exact case of CSP, for all distinct $\binom{2r}{r}/2$ permutations of labels. In case of the test for interactions, the procedures of Algorithm 3.5 are easily adapted by substituting the "for" cycle on the blocks with a cycle on all distinct pairs of blocks and computing the statistics accordingly.

In all-pairwise comparisons, the simultaneous confidence intervals are derived by the procedures in Algorithm 3.6 which is very similar to the one of Algorithm 3.4.

Rank-based tests

Conover (1999) gives an extension of the previously described Friedman test to the two-way design with replicates. Here, ranks are created within rows of the matrix of Figure 3.5 and, hence, go from 1 to rk. Denoted as R_i the sum of ranks for the i-th algorithm, $R_i = \sum_h \sum_t R_{hit}$, the test statistic to check the effect of algorithms in the general case with presence of ties is given by

$$S_F = \sum_{i=1}^{k} \frac{(k-1)}{k} \frac{[R_i - \mathrm{E}(R_i)]^2}{\mathrm{Var}(R_i)} \tag{3.18}$$

where $\mathrm{E}(R_i)$ and $\mathrm{Var}(R_i)$ are the population mean and variance which, in presence also of ties, are given by

$$\mathrm{E}(R_i) = \frac{br(rk+1)}{2} \qquad \mathrm{Var}(R_i) = \frac{r(k-1)}{k(rk-1)} \Big[\sum_{hit} R_{hit}^2 - \frac{rkb(rk+1)^2}{4} \Big]$$

If the design size is not too small, the distribution of S_F tends again to a χ^2 distribution with $k-1$ degrees of freedom.

Studying interactions with the instances by means of sums of ranks has been rarely attempted. Some references to works on this issue are given in Conover (1999) and Lehmann (1986). The Kendall's coefficient of concordance may be adapted to this case and employed to assess the degree of dependence of the ranked results on the instances.

[16] In the algorithm for the USP the implementation of the step to select ν_{ij}^* is still under revision. A tentative solution is given in the Appendix.

Procedure Compute_statistic_in_CSP();
$T^* = 0$;
generate a random permutation π of labels $\{1, \ldots, 2r\}$;
for all $k(k-1)/2$ distinct pairs (i, j) **do**
  for $h = 1, \ldots, b$ **do**
    Let $X_{pool} = X_{hi} \cup X_{hj}$ be the set of the $2r$ pooled observations;
    $X^*_{hi} = X_{pool}[\pi(1), \ldots, \pi(r)]$;
    $X^*_{hj} = X_{pool}[\pi(r+1), \ldots, \pi(2r)]$;
    $T^* = T^* + (\sum_t X_{iht} - \sum_t X_{jht})^2$;
  end
end

Procedure Compute_statistic_in_USP();
$T^* = 0$;
Select ν^*_{ij} from $\{0, 1, \ldots, r\}$ with a probability distribution that allows each permutation to appear with equal probability;
for all $k(k-1)/2$ distinct pairs (i, j) **do**
  for $h = 1, \ldots, b$ **do**
    $X^*_{hi} =$ shuffle X_{hi};
    $X^*_{hj} =$ shuffle X_{hj};
    swap X^*_{hit} with X^*_{hjt} for $t = 1, \ldots, \nu^*_{ij}$;
    $T^* = T^* + (\sum_t X_{iht} - \sum_t X_{jht})^2$;
  end
end

Algorithm 3.5: Synchronised permutations for a two way factorial design.

For all-pairwise comparisons, Conover (1999) gives the following formula to compute the minimal significant difference between means of ranks $\bar{R}_{.i.}$

$$MSD_F = t_{1-\alpha/2}\sqrt{\frac{2k(kr-1)Var(R_j)}{br^2(k-1)(bkr-k-b+1)}\left[1-\frac{T_F}{b(kr-1)}\right]} \tag{3.19}$$

where S_F is the statistic obtained by Equation 3.18 and the rejection of the general hypothesis through S_F is first required. However, both these tests on ranks can be obtained also by the very same procedures given for permutation tests, simply substituting the original data with their ranks.[17]

3.6.7. Remarks

Remark 1. If it is possible to control more than one factor in the determination of the algorithms, or a *stratification* of the instances is available, then we may consider a full factorial design. This arises, for example, when it is possible to characterise an algorithm by different parameters, or when instances are grouped, according to size or other features, in sub-groups called strata. The variables which identify the strata are not controlled, but they are supposed to have an effect on the experiment and their identification may be

[17] We observed that the confidence intervals produced by permutation tests with the scheme of Algorithm 3.5 on ranks are always slightly larger than those obtained by using Equation 3.19. Apparently, therefore, this algorithm is more conservative.

Procedure Compute_all-pairwise_MSD();
Choose an estimated error ϵ related with B;
Start: Choose a positive number MSD;
for all (i, j) of the $k(k-1)/2$ pairwise comparisons **do**
  if Compute_pairwise_MSD(i, j) returns false **then** goto Start;
end
Return MSD.

Procedure Compute_pairwise_MSD(i, j);

1. Subtract $\bar{X}_{hi.} - \bar{X}_{hj.} +$ MSD from every value of the data group relative to one of the two algorithms, say i, obtaining the new vector $X_{hit}(\text{MSD}) = X_{hit} - \bar{X}_{hi.} + \bar{X}_{hj.} - \text{MSD}$, $i = 1, 2, ..., r$. This vector is combined with the vector X_{hjt} and constitutes the pool of observations to permute.
2. Compute the statistic $T(\text{MSD})$ for the observed response $T(\text{MSD}) : T_0(\text{MSD}) = \sum_h \left(\bar{X}_{hi.}(\text{MSD}) - \bar{X}_{hj.}(\text{MSD})\right)$.
3. By rearranging the observations B times with *synchronised* permutations within the b blocks (see Algorithm 3.5) obtain the permutation distribution of the statistic $T^*(\text{MSD}) : \sum_h \left(\bar{X}^*_{hi.}(\text{MSD}) - \bar{X}^*_{hj.}(\text{MSD})\right)$.
4. Return "true" if the condition $|\#\{T^*(\text{MSD}) \leq T_0(\text{MSD})\}/B - \alpha_{PC}/2| < \epsilon/2$ is satisfied, else return "false".

Algorithm 3.6: An algorithm for computing simultaneous MSDs in the case of repeated measures. The confidence level $1 - \alpha_{PC}$ is set equal to $1 - \alpha_{FW}$.

important to reduce the variance of the results. Testing for interaction of strata variable is more meaningful than testing the interaction of single instances because whenever we are faced with a new instance, we would know to which class it belongs. A full factorial design is typically represented as in Figure 3.7. Let A and B be two factors (or strata) with respectively i and j levels and let b be the number of instances. For each combination of factor levels r runs are collected. The response X_{hijt} is the result obtained in a replicate t by the treatment combination i,j on the instance h.

The parametric tests for analysing a full factorial design are extensions from the two-way design. A series of rules that can be applied to derive each test statistic are reported in Dean and Voss, 1999, page 202. Common computer packages, like the R environment, have however these methods already implemented.

In full factorial designs, *interaction plots* are particularly helpful for the analysis. They are constructed by joining lines relative to levels of one factor at average response values of the levels of another factor and they are used to gain an idea of how different combinations of factor levels affect the responses. If the lines are not parallel, then a factor level performs differently with different levels of the other factor and an interaction may be hypothesised (see Chapter 4 for examples of such plots, *e.g.*, Figure 4.28).

All-pairwise comparisons depend on the presence or non-presence of interactions between factors. If there is no interaction, then pairwise comparisons between levels of one single factor may be considered. In the case of interactions, instead, the pairwise comparisons should better involve all treatment combinations. This second choice entails a higher number of comparisons and hence requires a higher adjustment of the α-value.

Extensions of permutations tests to full factorial designs are currently restricted to only cases with two levels per factors (Pesarin, 2001). Rank-based tests, instead, at the best of

Block (Instance)	Factor A	Factor B	Observations
1	1	1	$X_{1111}, X_{1112}, \ldots, X_{111r}$
		$\vdots$	$\vdots$
		k_B	$X_{11k_B1}, X_{11k_B2}, \ldots, X_{11k_Br}$
	2	1	$X_{2111}, X_{2112}, \ldots, X_{211r}$
		$\vdots$	$\vdots$
		k_B	$X_{21k_B1}, X_{21k_B2}, \ldots, X_{21k_Br}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
b	$\vdots$	$\vdots$	$\vdots$
	k_A	1	$X_{bk_A11}, X_{bk_A12}, \ldots, X_{bk_A1r}$
		$\vdots$	$\vdots$
		k_B	$X_{bk_Ak_B1}, X_{bk_Ak_B2}, \ldots, X_{bk_Ak_Br}$

Figure 3.7.: Full Factorial with complete blocking

our knowledge, have not been applied to full factorial analysis. In both cases, however, it is possible to consider all combinations of experimental factors and treat them as levels of one single factor, thus re-conducting the analysis within the known cases. Alternatively, it is possible to focus separately on each single factor and repeat the tests for each factor on the same data. In this case, the family-wise level of confidence must be adjusted for the multiple use of the same data.

Remark 2. So far, we only considered balanced cases, that is, cases in which the number of observations on an instance are the same for every treatment combination. Extensions to unbalanced cases are possible although they require the adjustment of the formulas. Yet, collecting the same number of runs per algorithm on all instances should not be a real impediment in experiments with algorithms.

However, an experiment can be unbalanced also on the strata of the blocks. For example, having as strata variable the size of the instance, it may occur that the number of instances available at each size is not the same. If instances come from real life applications or from benchmark libraries, this may be quite likely. Such a kind of unbalance may lead to a wrong inference because a large number of instances in one class in the analysis biases the results. If this effect is unwanted, because the sample is deemed not to be representative of the application, a possible kludge solution is given by the *bootstrap method*. In brief, a balanced design is bootstrapped from the original data preserving factorial combinations, that is, the observations in the ijk-th factorial combination of the bootstrap design are selected with replacement from the ijk-th factorial combination of the original unbalanced design. The resulting balanced design is analysed and the whole procedure repeated a number of times (indicatively 100 times should be enough). If the results are consistent with respect to the hypothesis tested in all the instances, then the corresponding conclusions may be drawn. If results are mixed, varying from bootstrap sample to bootstrap sample, then a larger number of instances should be collected in order to draw significant conclusions about interactions.

Remark 3. Stochastic algorithms for optimisation have commonly very asymmetric dis-

tributions of solutions quality. In the case of minimisation problems, this distribution is bounded from below, should have a left-end point as close as possible to the unknown minimum, and should be skewed to the right (*i.e.*, the lower end of the distribution is on the right). Therefore, the assumption of normality of data is often not appropriate and non-parametric tests are to be preferred.

The non-normality and asymmetry of the distributions also suggests that descriptive statistics such as the median and other quantiles are more appropriate for resuming, respectively, central tendency and variability. Sample mean and empirical variance are, indeed, efficient estimates only if the underlying distributions are close to normality or for sufficiently large sample sizes. Quantiles are, instead, preferable because they are scale-invariant for many basic transformations (da Fonseca et al., 2001; Sheskin, 2000).

Remark 4. Also the homoschedasticity of results among different algorithms seems arguable (consider, for example, the extreme case of a deterministic algorithm compared against a stochastic algorithm on one single instance). In this case not only the parametric tests presented are inappropriate but also the permutation tests, which require homoschedasticity for the exchangeability of data. Transforming data into ranks permits in this case the elimination of outliers, which are unwanted sources of variability within distributions, and rank-based tests appear therefore as the most robust among the methods presented, although they are also based on the assumption of homoschedasticity of the transformed data.

In all-pairwise comparisons with permutation tests, the simultaneous confidence intervals provided by Algorithms 3.4 and 3.6 are very conservative in the case of not homoschedasticity because they are largely affected by the algorithms with higher variance. An alternative procedure to compute the confidence intervals is however possible. It consists in using the procedure Compute_Pairwise_MSD with a confidence level $1 - \alpha$ adjusted by the Bonferroni's rule on each pairwise comparison, without then checking that the computed interval satisfies also the other pairs through the iterative procedure Compute_all-Pairwise_MSD. The advantage of this procedure is that the negative bias of an algorithm with an observed uncommon large variance remains confined to those comparisons involving the algorithm itself without affecting the other comparisons in which that specific algorithm is not involved.

Unfortunately, different MSDs for each pairwise comparison exclude the use of the graphical representation of results as shown in Figure 3.3. Hsu (1996) suggests an alternative representation, which allows to include confidence intervals of different length and maintains the unification of both practical and statistical information in a single plot. This plot is shown in Figure 3.8. It consists of a two dimensional space in which a $45°$ line represents the points satisfying $\bar{X}_i = \bar{X}_j$. At each point ($\bar{X}_i$,$\bar{X}_j$), representing the sample means of two algorithms, a segment is drawn of slope -1, centred in ($\bar{X}_i$,$\bar{X}_j$), and of length MSD$/\sqrt{2}$. Statistical inference is derived by checking whether the line segment crosses the $45°$ line. The practical assessment of mean differences is preserved, instead, on the x-axis or y-axis. All the $k(k-1)/2$ confidence intervals can be represented by drawing only the segments with $\bar{X}_i > \bar{X}_j$, *i.e.*, only intervals below the $45°$ line.

Remark 5. Clearly, the set of tests presented is not exhaustive. There may be other test statistics that are more appropriate in particular situations. Two tests, worth mentioning, compare two distributions in distinct ways.

- The *Binomial signed test* counts the number of positive and negative differences and uses the binomial distribution to test if the number of positive (or negative) differences is significantly different from an equal distribution. This test is appropriate

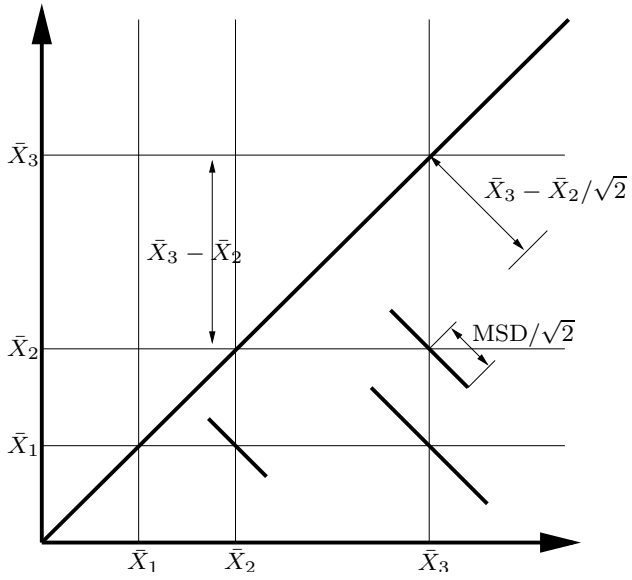


Figure 3.8.: Graphical representation of confidence intervals in a two dimensional space. This mean-mean scatter plot introduced by Hsu (1996) allows the representation of confidence intervals of different width.

to compare matched results of two algorithms when the only thing that matters is which one wins.

- The *Kolmogorov-Smirnov two-sample test* compares the empirical cumulative distribution function, CDF, of two samples. This test is able to detect more differences than all the tests previously introduced, because it is not based on mean values only. The test computes the maximal difference between the two curves, and exact quantiles or approximation quantiles for the distribution of this statistic are derived by permutation methods. Besides testing two algorithms, a variant of this test is also used to assess whether a sample comes from a known theoretical distribution.

In the rest of the thesis we conform to the experimental designs and the analysis here presented. In particular, we will design experiments according to design B and design C. The choice between the two will depend on the number of instances that are available. If the number is high, as in presence of a random generator, we will compare algorithms according to the scheme *"one single run on various instances"*; if the number of instances is not large we will collect several runs per algorithm on each instance according to the scheme *"several runs on various instances"*.

3.7. Sequential analysis

Often, in practical applications the time for developing and delivering a functioning algorithm is limited and a rigorous experimental design, as the one described in Section 3.6, might not be affordable. This kind of methodology imposes a well defined distinction between the development phase and the testing phase. Experiments planned beforehand

Procedure Race_Design_B;
repeat
  Randomly select an unseen instance and test all candidates on it;
  Perform *all-pairwise comparison* statistical tests;
  Drop all candidates that are significantly inferior to the best algorithm;
until only one candidate left or no more unseen instances;

Algorithm 3.7: Race for the experimental design *"one single run on various instances"*.

Procedure Race_Design_C;
repeat
  Run all candidates once on all instances;
  Perform *all-pairwise comparison* statistical tests;
  Drop all candidates that are significantly inferior to the best algorithm;
until only one candidate left or maximal number of runs exceeded;

Algorithm 3.8: Race for the experimental design *"several runs on various instances"*.

can require large computation time and necessarily force to restrict the attention to a limited number of candidate algorithms with hand tuned parameters and on small sets of benchmark instances. Moreover, in case the analysis of results unveils possible improvements for the algorithms (*e.g.*, by hybridisation) the whole process must be started anew. Clearly, one may wish to avoid unnecessary experimentations.

The issue of reducing the amount of experimentation is examined in statistics by the *sequential testing theory* which was firstly developed by Wald (1947) to minimise the cost of industrial testing. Currently, its application is fostered in clinical trail practice where, for ethical reasons, the expected number of patients exposed to a treatment has to be minimised. Recently, sequential testing has been proposed to solve the model selection problem in Machine Learning which consists, given a set of models and some training data, in determining the model that best describes the data (Maron and Moore, 1994, 1997). This problem of selecting the best among several alternative models has been recognised by Birattari et al. (2002) to be similar to the problem of selecting the best parameter setting for an SLS algorithm on a class of instances of a combinatorial optimisation problem given a set of training instances sampled from the class of interest.

In contrast to a brute force method, that corresponds to running all experiments planned beforehand, that is, all algorithms on all available instances, and choosing the best performing algorithm, the sequential analysis compares repetitively the available algorithms on a single instance, and ends when enough statistical evidence is gathered in favour of one algorithm.

There are three approaches for sequential testing: the original Wald's sequential probability ratio test, the Hoeffding race, and the testing for significance approach. For the first two we refer the interested reader to van der Tweel (2004) and (Maron and Moore, 1997), respectively. We focus, instead, on the latter approach which is the only one that has been used for the selection of a best algorithm in combinatorial optimisation by Birattari et al. (2002) and by Yuan and Gallagher (2004). In particular, we conform to the original procedure of Birattari et al. (2002) that received the name of "racing algorithm", as the best algorithm is the winner of the statistical comparisons against all other algorithms.

The original racing algorithm considers the *"one single run on various instances"* design and it is aimed at tuning SLS algorithms. A brief sketch is given in Algorithm 3.7. More in

detail, it works as follows. A set of instances is defined as the class of applicative interest. The elementary experimental unit of the race is the *stage* and it consists in running all candidate configurations of a metaheuristic on a single new instance. At the end of each stage a decision is made on which candidate should be discarded according to an all-pairwise test procedure. Four different statistical tests were compared for this task: three parametric tests based on all-pairwise *t*-test with α-value (i) adjusted by Bonferroni's rule, (ii) adjusted by Holm's procedure, or (iii) not adjusted, and one non-parametric test consisting of the protected Friedman test for all-pairwise comparisons (Equation 3.11). The simulation study conducted by Birattari (2004b) showed that the race based on Friedman test is the most powerful, that is, it yields the largest saving of experimental computation and no significant differences can be stated between the candidate selected by the race and the candidate that would result from a brute force approach, consisting in running all candidates on a number of instances equal to the instances used in the race until it was stopped.[18]

To maintain an overall type I error of α and to avoid inflation of the error rate, the tests at each stage should be performed at a lower experiment-wise value of α. This is, because the decision to maintain a candidate in the race depends on all previous decisions with multiple looks at the data. Lan and DeMets (1983) describe an adaptive way to "divide" α for a fixed number of stages according to the amount of information used. Note that in order to adjust the stage-wise α value and maintain an experiment-wise α-value, the maximal size of the experiment should be planned before starting the race. Apparently, however, given the result of the simulation study of Birattari (2004a), an adjustment of the α-value is not necessary in the specific context of his experiments. The issue on the adjustment is however quite controversial in the statistical literature and we believe that in the case of experiments on algorithms trading an increase of risk in committing a type I error for an increase of power is acceptable. Further increase of power can be attained by considering that an all-pairwise comparison procedure is not necessarily needed but a *comparison with the best* could suffice. However, to the best of our knowledge, only one parametric method exists for doing this (Hsu, 1996; Dean and Voss, 1999) while non-parametric solutions are needed in our case. Research on this issue would provide an important contribution to the amelioration of the racing algorithm.

The racing algorithm of Birattari can easily be extended to the *"several runs on various instances"* design. Although theoretical results show that this is not the ideal case for comparing optimisers (Birattari, 2004a), this is the highly probable context of real-life applications where the number of available test instances is usually small. In this case, the *stage* consists in running all candidate configurations of an SLS algorithm once on all the available instances. This extension is outlined in Procedure 3.8. The statistical analysis is based on the extension of the Friedman test to this experimental layout (Equations 3.18 and 3.19). We will use this modified version of the race for configuring an optimiser for the timetabling problem in Chapter 6.

We finally remark that the few applications of racing algorithms for the selection of best optimisers ended with one single winning candidate. If no control is applied on the α value, it is possible, at least in principle, to continue the race, generating new instances or collecting new runs, until enough evidence arises for selecting one single algorithm. But, if some algorithms are very similar, the race could become extremely long. A possible further development of racing algorithms, then, could consider a stronger involve-

[18] Birattari (2003) made available in the public domain an implementation of Procedure 3.7 for R, the free software suite for statistical computing. M. Birattari. CRAN - *Package* race. October 2004. `http://cran.r-project.org/src/contrib/Descriptions/race.html`. (October 2004)

ment of the null hypothesis for deciding that some algorithms are not different. In this way, the racing procedure could be faster because algorithms that seem to be identical in performance could be removed maintaining only one or because the race could stop even when high confidence is gained that algorithms are very similar. Unfortunately, while it is clear how this can be done with Wald's and Hoeffding's approaches, it is not clear how this could be done with the test for significance approach.

3.8. Time dependent analyses

The experimental designs and the statistical methods for their analyses presented in Section 3.6 are valid for a scenario, in which optimisation algorithms are given a strict time limit to find a solution. Solution quality is in this case the only stochastic variable which describes algorithm performance.

We already mentioned in Section 3.4 that a thorough description of an optimisation algorithm should take into account, besides solution quality also computation time (or *run time*). With stochastic algorithms, these two criteria are described by stochastic variables. In other terms the performance of a stochastic algorithm is characterised by a bivariate probability distribution. In this section, we outline the possible extensions in the analysis that arise when also this second variable is taken into account. We present the possible synthesis in a unified representation of both solution quality and run time. Two representations have been recently introduced by Hoos and Stützle (2004) and by da Fonseca et al. (2001) and it can be shown that they coincide. Yet, their statistical investigation remains still partly unsolved. We propose a further representation, which may inspire future research on this context. Then, we focus on the sub-case of fixed solution quality and discuss the possible insights on the behaviour of algorithms that can arise from the study of run time only.

3.8.1. Unified representation of time and quality performance

For a unified and general description of the performance of SLS algorithms in terms of time and quality, two models are possible. The implications arising from these models have only been marginally understood and applied. Further research in this direction is certainly needed. We present these two models pointing out their current use in the analysis of algorithms and giving reference for further possible developments.

Bivariate model. The behaviour of a stochastic optimisation algorithm can be modelled through a set of bivariate random variables X consisting in solution quality SQ and run time RT. If quality and time are judged equally important, the description of the algorithm must follow a bi-objective perspective. Each run of an algorithm generates a set of points $\mathcal{X} = \{X_i \in \mathbb{R}^2, i = 1, \ldots, M\}$, where the size M is random and the elements $X_i(SQ, RT)$ are the bivariate random variables. Each random element X_i is non-dominated in the Pareto sense[19] within the set. Finding a good descriptor of such a

[19] In a minimisation problem a point $X_1(RT_1, SQ_1) \in \mathbb{R}^2$ strictly dominates in Pareto sense a point $X_2(RT_2, SQ_2) \in \mathbb{R}^2$ if $RT_1 \leq RT_2$ and $SQ_1 \leq SQ_2$ and at least one of the two inequalities is strict. Throughout this section we use the term "quality" to indicate the "cost" of a solution according to the

set-characterised distribution is currently an active area of research (Coello et al., 2005). Da Fonseca et al. (2001) introduce the attainment function derived from random closed set theory as a simple and general way to describe the distribution of a set $\mathcal{X}$ of elements $X \in \mathbb{R}^d$ using the notion of goal-attainment. In our case, where $X \in \mathbb{R}^2$, the *attainment function* $\alpha_\chi(\cdot) : \mathbb{R}^2 \to [0,1]$ is defined as

$$\begin{aligned} \alpha_{\mathcal{X}}(z) &= P(X_1 \leq z \vee X_2 \vee \ldots \vee X_M \leq z) \\ &= P(\mathcal{X} \trianglelefteq z) \end{aligned}$$

The value $\alpha_{\mathcal{X}}$ indicating the probability that at least one element of $\mathcal{X}$ is smaller or equal (in Pareto sense) than $z \in \mathbb{R}^2$, that is, the probability of an SLS algorithm finding at least one solution of quality better than SQ_z within time RT_z in a single run. The attainment function can be see as the generalisation to random point sets of the multivariate cumulative distribution function $F_X(z) = P(X \leq z)$.

The attainment function is estimated via its empirical counterpart, in the same way as CDFs are estimated. This *empirical attainment function* is defined as

$$\widehat{\alpha}_n(z) = \frac{1}{n}\sum_{j=1}^{n} I\{\mathcal{X}_j \trianglelefteq z\}; \qquad I\{\mathcal{X}_j \trianglelefteq z\} = \begin{cases} 1 & \text{if} \quad X_j \trianglelefteq z; \\ 0 & \text{if} \quad X_j \triangleright z. \end{cases} \tag{3.20}$$

where each random set $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n$ corresponds to one of the n runs of the algorithm.

In simple words, $\widehat{\alpha}_n$ indicates the fraction of algorithm runs that produced at least one solution non-dominated by z. An example of the graphical representation of the empirical attainment function in the two dimensional case is given in Figure 3.9, left. For each value of α_n it is then possible to represent the boundary separating the points which are known to be attainable in at least a fraction $\widehat{\alpha}_n$ of the runs from those that are not. In a d-dimensional case this boundary corresponds to an *attainment surface*. In our two-dimensional case, the surface reduces to a curve in the plane time–quality. As in the univariate case the median value is a good descriptor of the empirical CDF, in the bivariate case the median attainment surface, corresponding to $\widehat{\alpha}_n = 0.5 = 50\%$, is a good descriptor for the empirical attainment function. In Figure 3.9, on the bottom right, we represent the median of an attainment function and its first and last quantile. Intuitively, they are the intersections of horizontal planes with the three dimensional curve. The computation of the empirical attainment function in two dimensions is related to the computation of the univariate empirical cumulative distribution function. But, whereas the latter must be computed only at data points the former is computationally more expensive, as it requires the computation at the data points and also at all the intersections of the data points.[20]

Empirical attainment functions of different algorithms can be compared statistically by means of Kolmogorov-Smirnov-like tests (Fonseca and Fleming, 1996). Yet, in case of the rejection of the hypothesis of no differences, the statement that one distribution is better than the other is possible only in presence of statistical dominance. If, instead, the empirical attainment functions cross, the comparison is inconclusive. In these cases, Paquete (2005) proposes a graphical representation of the probability differences of attainment functions $\widehat{\alpha}_{\mathcal{A}}(z) - \widehat{\alpha}_{\mathcal{B}}(z)$ between pairs of algorithms $\mathcal{A}$ and $\mathcal{B}$ for each point z in the plane. In this way, the amount of the differences can be visually inspected.

evaluation function and we focus on a minimisation problem. This explains the non-sense that lower quality is preferred over higher quality.

[20] An algorithm for doing this efficiently in the two-dimensional case was provided to the author by Carlos Fonseca (priv. comm.). An updated version has been made recently publically available online: `http://www.tik.ee.ethz.ch/pisa/` (May 2005).

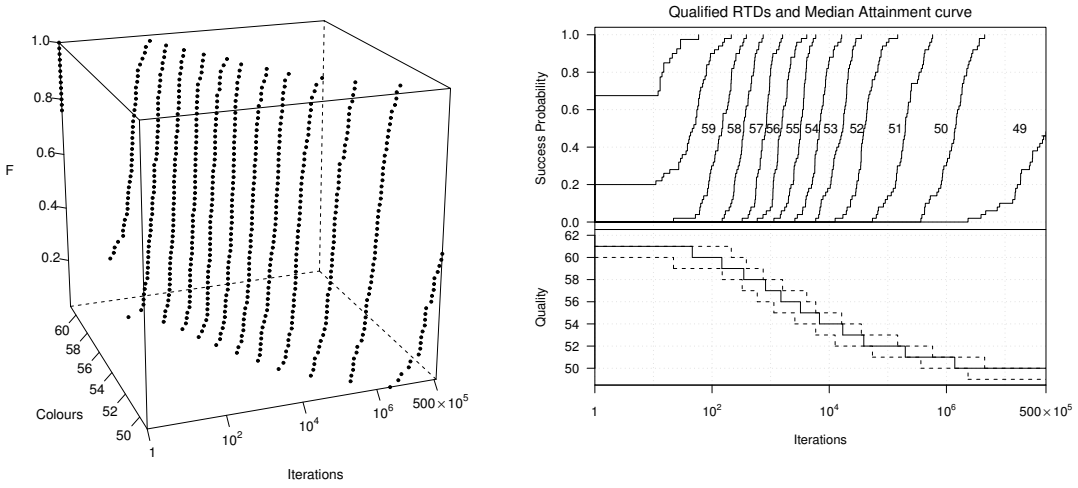


Figure 3.9.: On the left, a three dimensional representation of the empirical attainment function $\widehat{\alpha}$ or joint distribution function $\widehat{F}$. The data are collected from 50 runs of Tabu Search $\mathsf{TS}_{\mathcal{N}_1}$ for Graph Colouring on the instance `DSJC500.5` (see Chapter 4). The quality of the solution is expressed in the number of colours to minimise. The x-axis indicates the logarithm of the number of iterations. On the right, are represented the corresponding qualified run time distributions and empirical attainment curves. For the attainment curves, the median, the first and the last quantile are represented, derived from Equation 3.20 with $\widehat{\alpha} = 25/50$, $\widehat{\alpha} = 1/50$, $\widehat{\alpha} = 50/50$, respectively. The first and the last quantile, depicted with dashed lines, also correspond to the experimental lower and upper bounds of the empirical distribution. Note that at the start of the algorithm no solution is available, hence, in a non semi-logarithmic plot the attainment curve at iteration 0 should tend to infinity.

We remark that the attainment function in the two dimensional case corresponds in fact to the joint probability distribution $F_{SQ,RT}(q,t) = P[(RT \leq t) \cup (SQ \leq q)]$ used by Hoos and Stützle (2004) and called by them *run time distribution*. In the specific case of algorithm analysis, Hoos and Stützle (2004) observe that the empirical probability distribution can be computed by recording the run time and the solution quality whenever, during a single run of an algorithm, a solution is found with a quality strictly better than the best solution found so far in that run. More formally, if m is the number of runs and $sq(t,j)$ is the *trace* of the best solution quality found at time t in run j, the joint distribution function of $X_i(RT, SQ)$ is obtained as $\widehat{F}(RT \leq t \wedge SQ \leq q) = \#\{j | sq(t,j) \leq q\}/m$, which corresponds to counting for every point (q,t) the number of runs in which a solution of quality q is found within time t. The three dimensional plot of Figure 3.9 provides the representation also for $\widehat{F}(X)$. Yet, in practice, Hoos and Stützle (2004) confine themselves to the study of $\widehat{F}(X)$ for fixed values for one of the two variables. In the upper plot on the right of Figure 3.9, we represent the *qualified run time distributions* $\widehat{F}(RT \leq t \wedge SQ \leq \bar{q})$ that are obtained by sectioning the distribution function visualised on the left with planes parallel to the time axis, which in the Figure is expressed by the number of iterations. In an orthogonal way, it is possible to derive *solution quality distributions* by sectioning the distribution function with planes parallel to the quality axis. Hoos

and Stützle (2004) derive these sections, that is, the distributions $\widehat{F}(RT \leq t \wedge SQ \leq \bar{q})$ and $\widehat{F}(RT \leq \bar{t} \wedge SQ \leq q)$ by considering censored univariate cumulative distribution functions, (*e.g.*, $\widehat{F}(RT \leq t \wedge SQ \leq \bar{q}) = \#\{j|sq(t,j) \leq \bar{q}\}/m$.[21]

Both, in the notion of attainment functions and joint distribution functions the variables RT and SQ in each X_i can be dependent (as it actually is, given that longer run time guarantees better solution quality) but each variable X_i, is considered independent and each point of the plane time–quality receives an independent probability. How to estimate probabilities of attaining a whole set of points *simultaneously*, *i.e.*, all in a single run of the algorithm, if at all possible, remains unclear (Fonseca and Fleming, 1996).

Multivariate model. An alternative model, that received more attention in statistics and for which more methods of analysis are available, is the *repeated measurement design*. In this case, each algorithm is observed on a number of occasions during its run time. This yields for each individual algorithm repeated observations of its solution quality which are called *response profile*. Each response profile may be viewed as a multivariate variable $X_j(SQ(\tau_1), SQ(\tau_2), \ldots, SQ(\tau_t))$ where $\tau_1, \ldots, \tau_t$ are fixed time occasions and j is the algorithm's run number. The hypotheses being tested in such an experimental design are whether the algorithms have different response profiles over time. The assumption is that measurements are repeated a number of times on the same units (algorithm–instance) on fixed time occasions. Responses do depend on time but the time effect is not of primary importance. Note that the requirement of measuring the solution quality at fixed times is not restrictive, as it is in principle possible from the traces sq to derive the current best solution quality at each instant in time (the length of the intervals is arbitrary and can be determined by each instant in which an improvement occurred in at least one of the algorithms).

To better specify the design, we consider k algorithms running on one instance. For each of them we collect observations of the univariate variable $X(SQ)$. All algorithms are observed at t fixed time occasions $\tau_1, \ldots, \tau_t$ and the discrete stochastic process $\{X_{ij}(\tau_l), i = 1, \ldots, k, j = 1, \ldots, r, l = 1, \ldots, t\}$ is observed on each run j of algorithm i. A graphical representation of this process can be obtained by plotting at any time τ_l the median, or any other quantile, of $\bar{X}_{i.}(\tau_l)$ obtained from the r runs.

The analysis of such a model to determine whether algorithms are different at certain times, (*i.e.*, whether their response profile are different) is rarely being accomplished in the study of stochastic algorithms. Taillard (2001) proposes to apply a analysis at each point in time and to report where the test is significant. Although attractive, this procedure is not the most correct. Analogous analyses are attempted in biostatistics where the interest is in comparing the growth curves of treatments and diseases. Heitjan et al. (1993) review and compare statistical methods for the analysis of tumour growth experiments. The comparison is based on the type I error and on the statistical power. Results show that repeated tests at all times, or only at final time, exhibit type I error rate much higher than the declared experiment-wise α value. In contrast, methods that treat the series of data over time for each experiment as a single multivariate observation have contemporaneously more power and higher control over the type I error rate. The authors suggest MANOVA, Multivariate Curve Model, and regression with random effects and autoregressive errors as better methods for the analysis. Although in the specific case

[21] Note that the representation of $\widehat{F}(RT \leq t \wedge SQ \leq \bar{q})$ may also be obtained by $\widehat{F}(RT \leq t|SQ \leq \bar{q}) \cdot \widehat{F}(SQ \leq \bar{q})$ where the first factor is the non censured conditional distribution $\widehat{F}(RT \leq t \wedge SQ \leq q) = \#\{j|sq(t,j) \leq q\}/m'$ where m', $m' \leq m$, is the number of runs in which the quality $\bar{q}$ was attained, and the second multiplying factor is the probability of attaining this probability: $\widehat{F}(RT \leq q) = m'/m$.

of that article the last method is the best, the choice depends on the particular situation and on the validity of the corresponding assumptions. All three methods assume that treatments (in our case algorithms) are independent but that the observations within an algorithm are correlated. They also assume the normality and the homoschedasticity of the multivariate variable and, in the case of the last two methods, that data relative to each treatment can be fitted by the same kind of curve. The commonly used curve models are log-linear models or more complicated piecewise polynomial in case the curve is not monotone. Gompertz distributions that start log-linear and then flatten are also used in biostatistics. The validity of the two last assumptions in the case of algorithms must be checked because, as discussed above, the assumption of normality of the distribution of results at a certain time does not appear reasonable. Furthermore it must be verified if the data of all algorithms can satisfactorily be fitted by curves of the same type. If we consider, for example, simulated annealing and tabu search, it is very likely that the first algorithm due to its cooling scheduling has a curve which descends slowly at the beginning and faster after a certain time while tabu search has typically a much faster descent at the beginning. It is however reasonable to expect that all curves flatten at long run time to a limiting quality.

Among the different methods, however, MANOVA appears the most appropriate for our goals. It considers a multivariate linear model

$$X_{ij}(\tau_l) = \mu + \alpha_i(\tau_l) + \epsilon_{i,j}(tau_l)$$

where $\alpha_i(\tau_l)$ are the main algorithm effects and $\epsilon_{i,j}(\tau_l)$ are the error terms. Then, under the assumption of multivariate identically normally distributed, the Hotelling-Lawley trace test statistic (Arnold, 1981) can be used to determine an overall effect.[22] If the test is significant, one can proceed to univariate tests at each time; the global test through the Hotelling-Lawley guarantees that the type I error rate is preserved at the experiment-wise α. If the assumption of normality is not reasonably met, permutation tests may be used based on the same statistic or on invariant simplifications. Permutation tests, in which the original observations are replaced by ranks have also been defined (Good, 2000; Pesarin, 2001) and their application could be actually more appropriate for the analysis of algorithms.

We finally observe that in both these models, the bivariate and the multivariate model, the visualisation of results and a corresponding analysis is only possible if single instances are considered. To the best of our knowledge, no attempt has been done to generalise the analysis to sets of instances and it is not clear if this is possible. A generalisation of the multivariate model to more than one instance is possible but only if the times for solving them are comparable. In other cases, where the instances to solve are considerably different so that longer or shorter times are required for reaching comparable levels of normalised solution quality, the analysis should proceed separately through sub-groups of instances that are recognised to have similar characteristics.

[22] Differently from the Hotelling T^2, the Hotelling-Lawley test considers observations which are correlated. The test is available in R.

3.8.2. Qualified run time distributions

In order to focus on qualified run time distributions, a goal must be defined. In the case of optimisation problems with constraints, there are two kinds of goals that can be defined:

- the attainment of a feasible solution, that is, a solution that satisfies all hard constraints;
- the attainment of a feasible solution of a certain quality, that is, a solution that satisfies all hard constraints and whose evaluation function with respect to the soft constraints is such that $f(s) \leq \bar{q}$, where $\bar{q}$ defines the required value level of quality.

Although an analysis similar to the one described in Section 3.5 is perfectly suitable also when studying run time instead of quality, the interest in run time analysis for optimisation problems goes beyond the mere comparison of performance.

We distinguish four possible analyses based on run time distributions (RTD).

Comparison of algorithm performance. The comparison of algorithms to attain a feasible solution or a solution of a given quality can be done more comprehensively through RTDs curves. As noted in Section 3.6.7, the Kolmogorov-Smirnov test may be used to check for statistical significance of differences between RTDs.

Predicting algorithm termination or last improvement occurrence. It might be useful to investigate how RTDs for a specific algorithm vary when tighter solution quality is required. In this way, it might become evident that a certain level of solution quality can always be guaranteed in a given time limit or that the algorithm may profit from additional time. This latter situation is detected by the presence of truncated RTDs (*i.e.*, RTDs that do not reach the value one) if a given solution quality is not always attained before the time limit expires. In this case, techniques from survival analysis might be used to estimate time limits that guarantee with a high probability reaching a certain solution quality level. A strong limitation of this analysis, however, is that the definition of a quality level remains strongly instance dependent. A possible kludge, then, might be the study of RTDs of last improvements in algorithm runs. This curve may reveal whether the whole run time is useful or not.

In the case, in which the algorithm has a termination criterion other than a time limit, as, for example, a maximal number of iterations, a run time distribution might be defined for the run time necessary to reach termination. Hoos and Stützle (2004) denote these as *termination time distributions*. The graphical representation of these RTDs curves is useful for understanding the degree of variation of the algorithm's run time and the scaling effect due to different problem characteristics (we will use such an analysis in Chapter 4 and 5).

Summarising RTDs with theoretical distributions. Survival analysis techniques offer useful tools for the study of RTDs. Of particular interest is the approximation of empirical RTDs by means of theoretical distributions. The use of theoretical probability distributions to model empirical distributions favours the generalisation of results to new situations reducing the pitfall of over-fitting and makes possible inference on the behaviour of the algorithm suggested by the properties of the theoretical distribution.

The Kolmogorov goodness-of-fit test (Conover, 1999), based on cumulative distribution functions is used to test the null hypothesis that the empirical RTD is one of the family of a theoretical distribution function with parameters estimated from the data. The families of theoretical distributions which are usually considered for RTDs are the exponential, the Weibull, the log-normal, the gamma and the log-logistic.

An exponential run time distribution has the form $F(t) = 1 - e^{-\lambda(t)}$ where λ is the rate parameter. The exponential distribution has the property of being memoryless in the sense that the probability of finding a solution within a fixed time does not depend on the time passed so far and hence improvements occur randomly in time. In the study of algorithms, as observed on TSP and SAT problems (Stützle and Hoos, 2001; Hoos and Stützle, 2000), the comparison of an RTD with a fitted exponential distribution is useful to understand whether it is better leaving the algorithm running or restarting it after a given *cut-off time*. To this end it is checked whether the empirical RTD approximates an exponential distribution with $\lambda = 1/\bar{t}$, where $\bar{t}$ is the arithmetical mean of the empirical data. If this is not the case then one may proceed varying the inclination of the exponential distribution until a point is found where the steepness of the empirical RTD becomes and remains lower than the one of the exponential distribution. In that point it may be useful restarting the algorithm (we defer to Chapter 6 and to Chiarandini and Stützle, 2002 for an example of such analysis on a timetabling problem).

If the goal of the analysis is not determining possible cut-off points but simply describing the distribution through a theoretical one, the presence of an initialisation phase entails that an exponential distribution that better approximates the RTD has a translation of $\delta > 0$ such that $t' = t - \theta$ rather than t is used in its definition. The first attempt to estimate the parameters θ and λ can be done by setting again $\lambda = 1/\bar{t}$ and the value of θ equal to that of the first solution time. Then, varying the parameters probably a better fit can be found. The utility of finding a good fit is to possibly summarise the RTD by only two parameters, θ and λ. In case of truncated RTDs having a good fit through a theoretical distribution allows to predict the behaviour of the algorithm when longer run times are allowed.

If the exponential distribution does not provide a good fit, one can attempt to use the Weibull distribution which tends to better represent life data and is commonly used in reliability analysis. The Weibull cumulative distribution function is $F(t) = 1 - e^{-\frac{(t-\theta)}{b}^a}$ where a is the shape parameter, b the scale parameter and θ the location parameter. The presence of three parameters in the distribution makes the chance for good fits higher, but the estimation of the parameters from data harder. To estimate the three parameters of the Weibull distribution one can use non linear least squares regression techniques or, preferably, the more efficient maximal likelihood method (Venables and Ripley, 2002; Al-Fawzan, 2000).

Analysis on set of instances. The analyses in the previous paragraphs are possible only on single instances and their significance is therefore limited. It might be reasonable, then, trying to extend the results to a class of instances.

Qualified RTDs for a class of instances can be obtained by defining a quality measure that is invariant over the instances. Alternatively, other algorithm features, such as the above mentioned termination criterion or last improvement, can be investigated through aggregated RTDs. The RTDs correspond in this case to the empirical cumulative distribution function of the data pooled over several instances. As shown by Hoos and Stützle (1998) this procedure may however lead to wrong conclusions. A better procedure might be to approximate the RTDs on each instance with some theoretical distribution and then

use the corresponding parameters as representative measures. In a more simplistic way, the median value of each distribution may be used but truncated RTDs induce to use this approach with caution.

The first use of RTDs on a class of instances is observing the variability of instance hardness for a given goal, a level of solution quality or the feasibility of the solutions. Also interesting is verifying whether the chosen time limit guarantees limiting behaviour for the algorithms on all instances of the class or, in the presence of a termination condition, analysing the variation of halting time.

We remark, however, that the use of RTD on a set of instances is reasonable only when we assume that the instances induce a similar behaviour of the algorithm with respect to computation time. If the goal is classifying instances according to similar responses on the RTDs then techniques from exploratory data analysis may be useful. Correlation plots and regression analysis for instance scaling effects are examples of exploratory data analysis.

3.9. Landscape analysis

The methods for the analysis described in the previous sections are useful for assessing the performance of algorithms but do not provide explanations on the search mechanisms underlying these algorithms. SLS methods are intriguing because they perform well where other mathematical models have difficulties although they are based solely on intuitive rules whose validity is sometimes not clear. Particularly surprising is the effectiveness of local search which is in first place responsible for the success of such methods. One main concern is how to generalise its behaviour from some problems or from some specific instances.

Studies to improve the understanding of SLS algorithms based on local search have received increasing interest in recent years. The search space characteristics of a number of standard problems have been deeply investigated, including the Satisfiability Problem (Frank et al., 1997), the Travelling Salesman Problem (Merz and Freisleben, 2000), the Job Shop Scheduling Problem (Watson et al., 2003; Watson, 2003), the Linear Ordering Problem (Schiavinotto and Stützle, 2004), and others (Merz, 2000). These studies focus on various properties of the search space which might have an impact on the performance of SLS algorithms.

The search process of a *local-search-based* algorithm applied to a problem instance I can be seen as a walk on a *neighbourhood graph*, $G_I(\mathcal{S},\mathcal{N})$, induced by the neighbourhood structure $\mathcal{N}$ and the search space $\mathcal{S}$. In this neighbourhood graph, vertices correspond to solutions $s \in \mathcal{S}$ and edges connect neighbouring solutions. Typically, the neighbourhood is symmetric.

Definition 3.1 *A* search landscape $L_I(\mathcal{S},\mathcal{N},f)$ *for a problem instance* I *corresponds to the neighborhood graph* $G_I(\mathcal{S},\mathcal{N})$ *determined by the search space* $\mathcal{S}$ *and the neighbourhood structure* $\mathcal{N}$*, with a value* $f(s)$*,* $f : \mathcal{S} \to \mathbb{R}$*, associated to each vertex* $s \in \mathcal{S}$*. The value* $f(s)$ *is called vertex* level.

The definition of fitness landscape dates back to Wright (1932) who studied the roles of mutation, crossbreeding, and selection in evolutionary biology. In that context, there

is a space of possible genotypes, each genotype with a certain "fitness", and the distribution of fitness values over the space of genotypes determines the *fitness landscape*. In optimisation, the role of genotypes is taken by the possible solutions to the problem and the evaluation function determines their fitness (for an in-depth study of the similarities between biology and optimisation we refer the reader to Goldberg, 1989).

The search landscape can be visualised through similarities with the natural features of a land surface. It may be a region more or less mountainous, with many peaks of high level flanked by steep ridges and precipitous cliffs falling to profound valleys. For the rest of this work, it may be useful to define formally the following classes of points that may be identified in a search landscape.

Definition 3.2 *A* plateau of level z, $z \in \mathbb{R}$, *is a maximal connected subgraph* P *of* $L_I(\mathcal{S}, \mathcal{N}, f)$ *such that* $f(p) = z$ *for all* $p \in P$. *A plateau* P *is then:*

- *a* local minimum *if there does not exist any* $p \in P$ *and* $s : s \in \mathcal{N}(p)$ *such that* $f(s) < f(p)$;
- *a* bench *if there exists a* $p \in P$ *and a* $s : s \in \mathcal{N}(p)$ *such that* $f(s) < f(p)$. *In this case, the solution* s *receives the name of* plateau exit.

A study of the features of the search landscape, such as connectivity, solution density, distribution of local minima, ruggedness and plateau size, may be useful to (i) improve the current algorithm, (ii) explain the behaviour of SLS algorithms, (iii) understand the reasons why a problem is difficult or not, (iv) generalise the hardness of problem instances to larger instance classes, and (v) link *a priori* knowledge on the instance with characteristics of the search landscape and the consequent appropriate tuning of algorithm. More in general, the ultimate goal of landscape analysis is to create a general theory of local search based on the relationship between search landscape features and SLS performance.

Determining all features of a search landscape requires the exhaustive enumeration of all possible solutions. Alternatively, in some cases, where the search graph has some particular properties, it might be possible to characterise analytically some of the features of the landscape, as pointed out by (Grover, 1992) or (Dimitriou and Impagliazzo, 1996). Typically, however, problems from real world applications are complicated and such analyses infeasible. In these cases, the techniques are based on approximations derived by sampling or estimating landscape features through surrogate measures.

A variety of measures have been used in the literature for the approximate characterisation of the search space. We review some of the most important. However, what can be inferred from each single measures independently by the others is not clear. Validating or refuting intuitive relationships between landscape features and local search behaviour is still an open issue of landscape analysis.

Autocorrelation function. One important feature of the landscape is its "correlation structure", that is, how similar neighbouring solutions in the search space are with respect to the evaluation function. A *smooth* landscape is one in which neighbouring points in the space have similar levels. Knowing the level of one point carries a lot of information about the level of the neighbouring points and the situation is favourable to local search procedures which base their search on the local information about the evaluation function. At the opposite extreme, a *random* landscape is one in which the level of neighbouring points is entirely uncorrelated. Knowing the fitness at one point would then

carry no information about the fitness of neighbouring points and a local search based on the information of the evaluation function is expected to perform not differently from an uninformed random walk.

One possible way to determine the ruggedness of the landscape is by performing an explorative random walk in it. It starts from a randomly selected initial candidate solution, and at each step it goes to a randomly chosen neighbour; a walk in m steps results in a series of evaluation function values $(f_1, \ldots, f_m)$. In time series analysis, a measure to detect non-random trends in the data is the *(empirical) autocorrelation function* which is defined as

$$r(i) = \frac{\sum_{j=1}^{m-i}(f_j - \overline{f})(f_{j+i} - \overline{f})}{\sum_{j=1}^{m}(f_j - \overline{f})^2} \tag{3.21}$$

where $\overline{f} = 1/m \cdot \sum_{j=1}^{m} f_j$. In optimisation, the value $r(i)$ indicates the correlation between two points that are i steps apart in the random walk (Weinberger, 1990). Of main importance is $r(1)$ because it captures the statistical dependency between the level of a point in the landscape and its direct neighbours: $r(1)$ close to 1 corresponds to a smooth landscape, while a $r(1)$ close to 0 corresponds to a random landscape.

Clearly, the starting point has a strong influence on the character of the random walk. In order to generalise the results the random walk must, then, be repeated several times so that a sample of reasonable size of the search space is collected. If the values of $r(1)$ remain similar the information provided can be used to describe the whole search landscape, provided that we assume the landscape to be regular in every of its points and directions (*i.e.*, homogeneous and isotropic).

Descriptions beyond the first neighbourhood are, instead, rare in landscape analysis for optimisation (see Hordijk and Manderick, 1995 for the only example to our knowledge) but might as well be worth to investigate. Insights may be obtained by autocorrelation plots which are scatter plots of $r(i)$ for different distance values i.

For some problems, the correlation structure of the landscape can be determined analytically. Stadler (1996) derives analytical result for the correlation length, defined as $l = 1/|ln(r(1))|$, for some particular search landscape that satisfy a certain difference equation with respect to neighbouring solutions similar to wave equations in mathematical physics (Grover, 1992). Barnes et al. (2003) extend these results to a broader class of search landscapes. Some problems like travelling salesman, min-cut graph partitioning, graph colouring, and a version of the satisfiability problem have this property. In particular, graph colouring, with an evaluation function that measures the number of edges that connect vertices with the same colour, has $r(1) = 1 - 2k/(k-1)n$, where k is the number of colours and n the number of vertices (Stadler, 1996). This result entails that the correlation of nearest-neighbours depends only on the number of colours and the size of the graph. Hence, all instances of graph colouring solved at the same k have a similar search landscape and the behaviour of local search is expected to be similar. Nevertheless, this fact contrasts with the observation, reported in Chapter 4, that local search methods may encounter different difficulties when solving graphs with different structure. Apparently, the autocorrelation function is not enough to explain the behaviour of SLS algorithms.

Fitness distance analysis. *Fitness distance analysis* focuses on the relation between solution quality and solution distances (Jones and Forrest, 1995). The relation is summarised by the correlation coefficient and is also often represented graphically by *fitness-distance plots*. The solutions considered may be randomly sampled or, more frequently,

are local optima. In the analysis of SLS algorithms, the focus is usually on samples of locally optimal solutions. Particularly interesting is then the relation between local optima and global optima. If optimal solutions are not available best known solutions may be used in their place (in which cases the interpretation of results must be treated with caution). For minimisation problems, a large positive correlation coefficient indicates that the lower the evaluation function value, the closer the respective positions are, on average, to a globally optimal solution. A value close to zero indicates that the evaluation function does not provide much guidance towards globally optimal solutions, while for negative correlations, the evaluation function is actually misleading.

The results of fitness distance analysis have impact on the design of SLS algorithms. Indeed, highly correlated search landscapes suggest that the use of intensification strategies in SLS algorithms leads to good performance while strong diversification may be useful for the cases of weak correlation. Cases of negative correlations may instead suggest that the use of local search is not appealing and that different solution methods should be considered. Moreover, fitness-distance correlation might also be used to evaluate different neighbourhoods, giving preference to those that allow higher correlation.

Fitness distance correlation alone, however, is never enough to account for differences in the difficulty of individual instances. Moreover, discordant comments on its interpretation are reported in the literature (see Naudts and Kallel, 2000 for a discussion on the limitations of this approach).

Local optima localisation. The notion of local optima is crucial to SLS algorithms. SLS algorithms search for local optima, explore them and try to exit from them. Characterising the location of local optima in the search space is, therefore, useful to unveil the difficulties inherent in a search landscape. A sample of local optima may be obtained by a relatively simple SLS algorithm. Two distributions of distances may then be considered: the pairwise distances within the set of local optima, or the distances between local optima and closest optimal solutions. In both cases, the range of observed distances and the modes of the distribution reflect important properties of the relative placement of local optima across the landscape. For example, a multi-modal distribution of pairwise local optima distances reveals the concentration of local minima in a number of clusters where the lower modes correspond to the intra-cluster distances and the higher modes represent the inter-cluster distances (see Paquete et al., 2004). In some other cases it was shown that the average distance of local optima from the nearest optimal solution gives rise to the most significant model to predict the hardness of an instance (Watson et al., 2003).

Clearly, the crucial step in this analysis is computing the correct distances between solutions. For many neighbourhoods and solution representations the problem of finding the minimal distance between two solutions given a move operator is an $\mathcal{NP}$-hard problem and approximate measures may become necessary. The indications provided by the approximate measures are then reliable only if highly correlated with the exact value. However, (Schiavinotto and Stützle, 2005) observe that for some problems represented by permutations the right distance value can be computed efficiently, and that the use of an approximate value is unjustified. In addition to this, it must be observed that distances between pairs of solutions peak in a value which is typical of the move operator and the analysis of results should distinguish observations determined by the algorithm from those which are a typical effect of random sampling the search landscape.

Number of local and global optima. Another characteristic of local optima that could be taken into account is their number. It is reasonable to assume a negative correlation between the number of local optima and the hardness of solving a problem instance by local search. Indeed, in the extreme case that all local optima are global optima, local search would always find a solution to the problem while, in all other cases, the chances that local search ends in a local optimum that is not a global one increases with the number of local optima.

In general, neighbourhoods, that entail fewer local optima or local optima of better quality should be preferred. The analysis of these two features, possibly even from a theoretical point of view, may be relevant for the selection of the neighbourhood structure. However, when more complex SLS algorithms are introduced the impact of the number of local optima on this issue becomes less clear.

Finally, the number of global optima in the search space is also indicative of the effort for local search to locate them. Intuitively, the lower this number is, the harder it should be to reach optimal solutions. In contrast, with many optimal solutions spread over the search landscape, random restart algorithms are likely to perform well.

Backbone size. The backbone of a problem instance is the set of solution components that maintain identical values in all optimal solutions of the instance (Monasson et al., 1999). The fraction of solution components appearing in the backbone determines the backbone size. Studies, above all on the SAT problem, showed that the backbone size is correlated to the search cost of locating solutions. The SAT problem is known to exhibit a sharp transition from satisfiable to unsatisfiable problems, and this phase transition can well be characterised by a few features of the instances. It has been noted that there is a common pattern of problems easy-hard-easy in correspondence of this phase transition and this has stimulated researchers to investigate the reasons why instances become harder. It has been noted that the backbones rapidly pass from small to large size close to that region, but contemporaneously the number of solutions drastically decreases (Achlioptas et al., 2000). The backbone size appears, therefore, as a redundant information on the number of optimal solutions. This conjecture was confirmed on the Job Shop Scheduling problem by Watson et al. (2003).

Plateaux. In order to escape from a plateau, a local search procedure has either to find an exit or to accept a move to a worse neighbour. The characterisation of the plateaux, that is, the definition of the fraction of plateaux that are benches or local minima might, therefore, help to decide which strategy to adopt. Intuitively, however, the larger a plateau is the more costly it is to find an exit. The size of plateaux, therefore, has an impact on the design of SLS algorithms. With small plateaux it might be worth to spend some iterations moving in the plateau in search of an exit, with large plateaux, instead, it might be better to diversify soon the search jumping to other regions of the search space.

To determine its size, the plateau has to be exhaustively explored. In this case sampling does not help and exploration of plateaux may be achieved with standard search techniques, introduced in Section 2.3.3.

Connectivity. A property of the search landscape which is of chief importance for local search, above all in highly constrained problems, is the connectivity of the search landscape.

Definition 3.3 *A landscape $L = (\mathcal{S}, \mathcal{N}, f)$ is* connected at a level l *if, and only if, there exists a path in the neighbourhood graph $G_{\mathcal{N}}$ between any two solutions $s, s' \in \mathcal{S}$, visiting only solutions t with $f(t) \leq l$. If no restriction on $f(t)$ is given, we that L is* connected*, without further specifications on the level.*

The neighbourhood definition plays a fundamental role in the definition of connected landscapes. Most neighbourhood structures result in a connected landscape, if we do not impose any restriction on the level. But things may change with the introduction of constraints and with the adoption of particular search strategies.

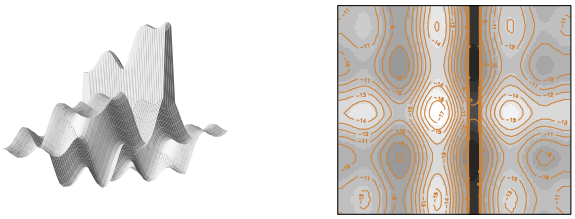
We illustrate the possible pitfall with an example. In general, search landscapes are multi-dimensional and extremely difficult to visualise. In Figure 3.10 we reduce the search space to two dimensions and provide a three dimensional view which includes the evaluation function level and the contours of quality levels. A landscape similar to the one in Figure 3.10(a) may occur in a highly constrained problem with hard and soft constraints weighted in a single evaluation function. The higher barrier may then consist of all those solutions that violate some hard constraints, since these are much more severely weighted in the evaluation function. The search space is in this case clearly connected if no level restriction is imposed on the local search.

If we adopt a different strategy, that is, if we decide not to weigh all the constraints in the evaluation function but only the soft constraints and to forbid visiting infeasible solutions, then the search landscape becomes disconnected even without any level restriction on the local search. This case is visualised in Figure 3.10(b). Clearly, in a disconnected landscape the search may become trapped in single separated regions. In addition to this, SLS methods tend to limit the entity of worsening steps to jump out from local optima. A typical case is Simulated Annealing that uses a probabilistic acceptance criterion that links the probability of moving from a solution to a worse one to the size of the barrier between the two solutions in the landscape. As the search proceeds further and the probabilistic criterion becomes tighter, even small barriers become insurmountable. Thus the search landscape may become characterised by many small basins of attraction as depicted in Figure 3.10(c).

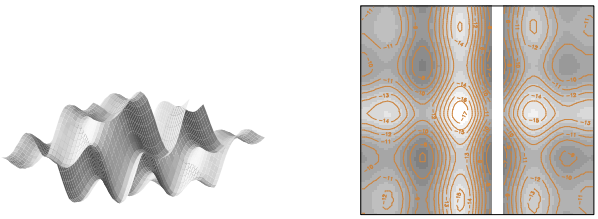
3.10. Discussion

We reviewed the main methods for analysing stochastic optimisers. We tried to collect and organise the tools of applied statistics that are most relevant in the context of combinatorial optimisation. In this process, we recognised that a statistical analysis of algorithms leaves many issues open. We will try to collect them in the conclusions of this thesis.

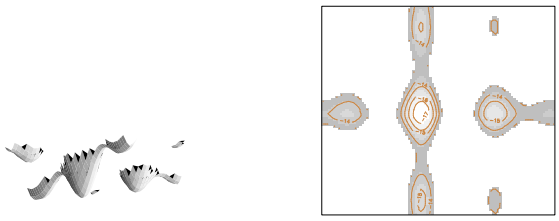
Of central importance for the whole thesis is the definition of the statistical methods for the analysis of experiments on the basis of solution quality. We will provide evidence in Chapter 4 of the difficulty to adapt parametric methods to the analysis of stochastic optimisers. We will instead compare permutation tests and rank-based tests and give preference to the most powerful. A simulation study is reported in Appendix B for validating and comparing these two approaches. When the behaviour of the tests is similar, the choice of the test depends on the real application because rank based tests remove the entity of the differences on single instances while permutation tests take these entities into account. However, the experience in this thesis seems to confirm the wide belief that parametric tests remain robust under considerable violations of some of their



(a) All constraints weighted in the evaluation function.



(b) The path through solutions that violate some constraints is not allowed (vertical white line).



(c) Only small deteriorations of solution quality are allowed.

Figure 3.10.: A simplified two dimensional search space landscape with a continuous evaluation function for three different SLS strategies.

assumptions, while instead the implementation of permutation tests still requires calibration efforts.

Of particular relevance is also sequential analysis that constitutes the central aspect in the experimental methodology for the development of SLS algorithms described in Chapter 6. A time dependent analysis will be used only to corroborate, or to restrict the extent of the results inferred by the previous methods.

The space devoted to landscape analysis in this thesis, instead, is marginal and certainly insufficient. This is a very attractive area of research, as it could unveil the mystery connected with local search algorithms and provide rational arguments for their use. But it certainly requires full time research and strong skills in a series of disciplines such as Mathematics, Programming, Statistics, Probability Theory, and Physics plus a certain degree of creativity in envisaging new hypothesis to test.

Chapter 4.

Graph Colouring

In which we deal with Stochastic Local Search algorithms for solving large instances of the graph colouring problem. We introduce new methods, study in-depth the use of large scale neighbourhoods, and provide extensive experimental results on benchmark graphs.

4.1. Introduction

The interest in graph colouring originates from the question about how many different colours are necessary to colour the countries of a map in such a way that no pair of adjacent countries receives the same colour. This question, that can be formalised by the use of planar graphs, has spawned an enormous amount of mathematical research and only rather recently the conjecture that four colours are enough for colouring planar graphs has been finally proved true Appel et al. (1977).

The graph colouring problem (GCP) for more general graphs than planar ones remains a central problem of graph theory with many important real life applications. Some of such applications were already mentioned in the introduction of this thesis and are register allocation (Allen et al., 2002), job scheduling (Leighton, 1979), air traffic flow management (Barnier and Brisset, 2002), light wavelengths assignment in optical networks (Zymolka et al., 2003), and timetabling (de Werra, 1985). Another real life situation that can be modelled as a GCP is testing for unintended short circuits in printed board circuits, where "nets" can be partitioned in supernets that can be tested simultaneously thus speeding up the testing process (Garey et al., 1976). Two applications of graph colouring have also been pointed out in Mathematics: solving the algebraic structure of Quasigroups (Gomes and Shmoys, 2002), and numerical estimation of large, sparse Jacobian matrices (Hossain and Steihaug, 2002). In statistics, the problem of constructing Latin square designs is also solved through graph colouring (Lewandowski and Condon, 1996, see also footnote 2 of Chapter 3, on page 45). All these applications entail solving large GCP instances where the size of the graphs usually exceeds 100 vertices.

In this chapter, we study the application of SLS methods for the GCP. We introduce a new local search with a very large scale neighbourhood, and we are the first to implement and test other variants of SLS methods that were successful for the satisfiability problem in propositional logic and for constraint satisfaction problems. We empirically assess the performance of construction heuristics, iterative improvement procedures, and high-performing SLS algorithms. In this latter case, we compare our new algorithms with state-of-art algorithms through a rigorous experimental analysis. The final outcome is an unbiased evaluation of approximate algorithms for solving large GCP instances which

will become an important reference in the empirical analysis and choice of algorithms for this problem. We also attempt a more detailed map of algorithms in relation to structural characteristics of the graphs to be solved. Finally, for some algorithms we provide more in-depth analyses showing the behaviour with respect to run time, investigating reasonable stopping times, or, explaining the reason why they perform unexpectedly poorly.

4.2. Formal definition of the problem and notation

In the graph colouring problem we are given an undirected graph $G = (V, E)$ and a set of colours Γ. The finite set V is the set of *vertices*, while the set $E \subset V \times V$ is the set of *edges*.

A *colouring* is a *mapping* $\varphi : V \mapsto \Gamma$ that assigns a unique colour to each vertex. The set of colours is written as a set of natural numbers: $\Gamma = \{1, \ldots, k\}$, and, hence $|\Gamma| = k$. Equivalently, a colouring can be seen as a *partition* $\mathcal{C}$ of the set of vertices into k subsets, that is, $\mathcal{C} = \{C_1, \ldots, C_k\}$, where the vertices in the *colour class* C_i are coloured with colour $i \in \Gamma$. The assignment φ and the partition $\mathcal{C}$ are two equivalent ways of defining a colouring and we will use either of them interchangeably as is convenient in the exposition.

A colouring is said to be *feasible* (or legal) if there are no pairs of vertices $u, v \in V$ such that $(u, v) \in E$ and $\varphi(u) \neq \varphi(v)$. Similarly, we say that a colouring is *infeasible* if there exists an edge $(u, v) \in E$ such that $\varphi(u) = \varphi(v)$. In this case we say that the end vertices of such an edge are *in conflict*.

The decision version of the GCP, called the *vertex k-colouring problem*, consists in finding a feasible colouring using a defined number k of colours. It can formally be defined as

> ***Input:*** *An undirected graph $G = (V, E)$ and a set of colours Γ with $|\Gamma| = k \leq |V|$.*
>
> ***Question:*** *Is there a k-colouring $\varphi : V \to \Gamma$ such that $\varphi(u) \neq \varphi(v)$ for all $(u, v) \in E$?*

The *chromatic number* χ_G is a characteristic of the graph and corresponds to the smallest k such that a feasible k-colouring exists. The optimisation version of GCP, also known as the *chromatic number problem*, consists in determining χ_G, and can be formalised as

> ***Input:*** *An undirected graph $G = (V, E)$ and a set of colours Γ with $|\Gamma| = k \leq |V|$.*
>
> ***Question:*** *Which is the smallest k such that a feasible k-colouring exists?*

The chromatic number problem can be approached by solving a decreasing sequence of k-colouring problems until for some k a feasible colouring cannot be found. In this case, the best feasible colouring uses $k + 1$ colours and this is the chromatic number of the graph.

Next, we introduce a few definitions from graph theory which will be used throughout the rest of the chapter. The *order* of a graph is the number of vertices in the graph, *i.e.*, $n = |V|$. The *edge density* $\rho(G)$ is the proportion of $|E|$ with respect to the potential edges of G, *i.e.*, $\rho(G) = |E|/\binom{|V|}{2}$. Graphs with a number of edges that is roughly quadratic in the number of vertices are usually called *dense*, as opposed to *sparse* graphs, which exhibit, instead, a linear dependence.[1]

[1]Note that, the notion of dense and sparse graphs makes sense only for families of graphs whose order

The *degree* of a vertex, denoted by $d(v)$, is the number of edges which have v as one of their end points. The maximal degree $\Delta(G)$ and the minimal degree $\delta(G)$ are, respectively, the maximal and the minimal vertex degree over all vertices in V. The *average degree* of G is given by

$$d(G) = \frac{1}{|V|} \sum_{v \in V} d(v).$$

If all the vertices of G are pairwise adjacent, then G is *complete*, while a set of vertices is *independent* (or *stable*) if no two of its elements are adjacent. An *independent* set is also denoted as a *proper* set, while if one, or more edges, exist between its vertices then it is said *improper*. A *path* is a non-empty graph $P_{l-1} = (V, E)$ with $V = \{v_0, v_1, \ldots, v_l\}$ and $E = \{(v_0, v_1), (v_1, v_2), \ldots, (v_{l-1}, v_l)\}$, where all v_i are distinct. A *cycle* is a graph $C_l = (V, E \cup \{(v_l, v_0)\})$ with V, E taken from the path definition . The length of a path or cycle is the number of edges it contains, *i.e.*, $l - 1$ or l, respectively. A *subgraph* of $G(V, E)$ is a graph $G'(V', E')$ with $V' \subseteq V$ and $E' \subseteq E$. A non-empty graph G is called *connected* if any two of its vertices are linked by a path in G. A *connected component* is a maximal connected subgraph of G. If all vertices of G are pairwise adjacent then the graph is *complete*. A complete graph on r vertices is denoted as K_r. The greatest number r such that a K_r is a complete subgraph of G with order r is called the *clique number* of the graph and it is denoted by $\omega(G)$.

In order to distinguish the value of the chromatic number and the clique number from approximations of their values we sometimes use for the approximations the notation $\widehat{\chi}(G)$ and $\widehat{\omega}(G)$, respectively. Clearly, we have $\widehat{\chi}(G) \geq \chi(G)$ and $\widehat{\omega}(G) \leq \omega(G)$.

We also introduce the following notation which will serve for the description of our algorithms (U and T are two subsets of V, and, for brevity, if $U = \{v\}$ we write $U = v$).

- V^c is the set of vertices in V that are involved in at least one conflict, *i.e.*, $V^c = \{v \in V : \exists u \in V, \varphi(v) = \varphi(u), (u, v) \in E\}$;
- $A_U(v)$ is the set of vertices in U adjacent to the vertex v, *i.e.*, $A_U(v) = \{u \in U : (u, v) \in E\}$;[2]
- $E_U(T)$ with $T \subseteq U$ is the set of edges that connects vertices in T with vertices in U, *i.e.*, $E_U(T) = \{(u, v) \in E : u \in U, v \in T\}$. Note that because of $T \subseteq U$, $E_U(T)$ contains also all edges (u, v) with u, v in T. As a special case, we denote with E_U^c the set of edges between vertices in U, that have the end vertices in the same colour class, *i.e.*, $E_U^c = \{(u, v) \in E : u, v \in U,\ \varphi(u) = \varphi(v)\}$. Clearly, $E_V^c = \bigcup_{i=1}^{k} E_{C_i}^c$.

The following relations between adjacent vertices $|A_U(v)|$ and incident edges $|E_U(T)|$ are derived trivially from the definitions above and by the principle of set theory that avoids multiplicity of the objects collected.

Remark 4.1 $|E_U(v)| = |A_U(v)|, \forall v \in U, U \subseteq V$.

tends to infinity. In addition, the concept of a sparse graph is distinct from the concept of low-density graphs. By sparse we mean families of graphs for which the number of edges $|E|$ is in $O(|V|)$ and so the density decreases with increasing $|V|$. Meanwhile, by low-density graphs we mean families of graphs which have $O(|V|^2)$ edges but for which the density, remains small but constant with increasing $|V|$ (for example we will encounter uniform randomly generated graphs which maintain a fixed density of around 0.1 independently from their size).

[2] In Graph Theory, the set A is often referred to as the neighbourhood of vertex v. Here we avoid this terminology in order not to confuse it with the notion of neighbourhood in local search.

Remark 4.2 $|E_U(U)| = |\bigcup_{v \in U} E_U(v)| = \frac{1}{2} \sum_{v \in U} |E_U(v)|, \forall U \subseteq V.$

4.3. Known theoretical results, complexity, and approximations

The focus of our study is on finding the chromatic number. For some special graphs,[3] $\chi(G)$ is known; such cases are the following ones.

- $\chi(G) = 1$ iff G is totally disconnected.
- $\chi(K_n) = n$;
- $\chi(C_n) = \begin{cases} 3 & \text{for } n \text{ odd} \\ 2 & \text{for } n \text{ even} \end{cases}$ and $n > 1$;
- $\chi(\overline{C}_{2n+1}) = n + 1$ where $\overline{C}_n$ is the complement (*i.e.*, a graph on the same set of vertices but with all edges not present in the original graph) of an odd cycle of order at least 5;
- $\chi(S_n) = 2$ where S_n is a star graph (*i.e.*, a tree with one vertex of degree n and all others of degree 1) and $n > 1$;
- $\chi(W_n) = \begin{cases} 3 & \text{for } n \text{ odd} \\ 4 & \text{for } n \text{ even} \end{cases}$ where W_n is a wheel graph and $n > 2$;
- $\chi(G) \geq 3$ iff G has a cycle of odd length;
- $\chi(G) \leq 4$, for any planar graph G. This famous result, called the Four Colour Theorem, seems to have been first conjectured in a letter from De Morgan to Hamilton in 1852. Recently, shorter proofs, yet still based on computer, appeared by Robertson et al. (1996).[4]

The k-colouring number problem for arbitrary k was shown to be $\mathcal{NP}$-complete by Karp (1972). The k-colouring problem is solvable in polynomial time only for $k = 2$, and for arbitrary k on the following special graphs: comparability graphs, chordal graphs, circular arc graphs, (3,1) graphs, and interval graphs (Garey and Johnson, 1979). The problem remains $\mathcal{NP}$-complete for $k = 3$, and graphs having no vertex degree exceeding 4 (Stockmeyer, 1973), and for arbitrary k on intersection graphs for straight line segments in the plane and on circle graphs.

[3] A formal definition of the special graphs mentioned in this section is not important in our discussion. The interested reader is referred to Diestel (2000) or, for a comprehensive survey on the topic, to Brandstädt et al. (1999).

[4] Cahit (2004) has recently proposed a non-computer proof but his paper is not yet published and a history of incorrect "proofs" for this problem suggests caution. Note that an $\mathcal{O}(n^2)$ time complexity algorithm for four-colouring planar graphs can be derived from the proof of Robertson et al. (1996) but it seems not practical with regard to implementation. With regard to our introductory problem of colouring geographic maps, in practice the need arises for models with more general graphs than the planar ones. Countries with exclaves like Russia and single point bordering countries make the four-colour theorem not applicable. In practice, the assignment task is solved by simple construction heuristics such as the DSATUR, described later (Freimer, 2000).

Approximability results		Non approximability results		
Factor	Due to	Factor	Due to	Assumption
$\mathcal{O}(n(\log\log n)^2/\log^3 n)$	Halldórsson (1993)	$\mathcal{O}(n^{1-\epsilon})$	Feige and Kilian (1998)	$\mathcal{NP} \neq \mathcal{ZPP}$
$\mathcal{O}(n(\log\log n/\log n)^3)$	Berger and Rompel (1990)	$\mathcal{O}(n^{\frac{1}{5}-\epsilon})$	Bellare et al. (1998)	$\mathcal{NP} \neq co\mathcal{RP}$
$\mathcal{O}(n(\log\log n/\log n)^2)$	Wigderson (1983)	$\mathcal{O}(n^{\frac{1}{7}-\epsilon})$	Feige and Kilian (1998)	$\mathcal{P} \neq \mathcal{NP}$
$\mathcal{O}(n/\log n)$	Johnson (1974)	$\mathcal{O}(n^{\frac{1}{5}-\epsilon})$	Bellare et al. (1998)	$\mathcal{P} \neq \mathcal{NP}$

Table 4.1.: Bounds on the approximability and non approximability of the chromatic number problem. Strength of the results increases as going upwards in the table. The two conditions $\mathcal{NP} \not\subseteq \mathcal{ZPP}$ and $\mathcal{NP} \not\subseteq co\mathcal{RP}$ are known to be equivalent.

"Bad" results are known even on the approximability of the chromatic number. For most existing polynomial-time graph colouring algorithms the absolute performance ratio can be as bad as $\mathcal{O}(n)$ where $n = |V|$ (we recall that the absolute performance ratio is the largest ratio of colours used by the algorithm to the chromatic number over all possible graphs, see also Section 2.2). The best absolute performance guarantee of $\mathcal{O}(n(\log\log n)^2/\log^3 n)$ for an approximation algorithm is given by Halldórsson (1993).

Even worse, there exist results that show that, unless $\mathcal{P} = \mathcal{NP}$, no polynomial time approximation scheme may exist for certain approximation ratios. We report some of these results in Table 4.1. The tightest bound on polynomial time approximation schemes for the chromatic number is presented by Feige and Kilian (1998) who state that unless $\mathcal{NP} \subseteq \mathcal{ZPP}$ it is intractable to approximate $\chi(G)$ to within $n^{1-\epsilon}$ for any constant $\epsilon > 0$. The class $\mathcal{ZPP}$, $\mathcal{P} \subseteq \mathcal{ZPP}$, arises in the theory of randomised computation and comprises problems that can be solved in polynomial time by a probabilistic algorithm that makes no errors. More details on approximability for graph colouring can be found in the survey by Paschos (2003) and on the Internet (Crescenzi and Viggo, 2004).

In the general cases lower and upper bounds on the chromatic number exist. A simple upper bound is given by $\Delta(G) + 1$. Brook's theorem tightens this bound to $\chi(G) \leq \Delta(G)$ if G is neither complete nor it has an odd cycle. A lower bound is instead ω_G. However finding the maximal clique in a graph is also an $\mathcal{NP}$-hard problem (Garey and Johnson, 1979). Tighter lower bounds for specific classes of graphs may be found in the literature. Johri and Matula (1982) provide tables on lower bounds for random graphs obtained with the probabilistic method, which is a versatile approach for graph theory on random graphs. They also provide tables for estimates on the chromatic number. Finally, a conjecture based on a theorem of Bollobás and Thomason (1985) states that, for random graphs of sufficiently large order and with edge density equal to 0.5, $\chi(G)$ is with high probability equal to $|V|/2\log_2|V|$ (Bollobás, 2004).[5]

It seems reasonable to speculate that graphs with a large chromatic number must have large cliques and hence small girth (*i.e.*, the length of the shortest cycle in a graph). Yet, this may not be true. Erdős (1961) showed that for any two positive integers g and k

[5]The complete formula would be $\frac{n\log(1/q)}{2\log n}(1 + o(1))$ where $q = 1 - p$, with p being the edge density, and $o(1)$ denotes a function of n converging to zero as $n \to \infty$. For graphs of order 1000 and edge density 0.5, $o(1)$ is still about 0.6 as shown in Bollobás and Thomason (1985), hence the approximation given in the text is actually valid only for graphs of order much larger than those to which we will address our attention. The probabilistic approximations provided by Bollobás and Thomason (1985) and Johri and Matula (1982) are very close as, for example, for $n = 1000$ and $p = 0.5$ they indicate $\chi(G) \approx 80$ and $\chi(G) \approx 85$, respectively. The tables in Johri and Matula (1982) present, however, more detailed results. Other improved probabilistic results are those of Luczak (1991) and Achlioptas and Naor (2005). The latter one states that the chromatic number for a random graph G_{np} with $p = d/n$ is either k_d or $k_d + 1$ where k_d is the largest integer k such that $d < 2k\log k$. For $n = 1000$ and $p = 0.5$ this entails a $\chi(G)$ of 59 or 60. Yet, these values are stronger, as not obtained by computer simulation.

there exists a graph of girth at least g with chromatic number at least k. This important result suggests that a large chromatic number is not always caused by some dense local substructures, which could be easily detectable, but rather it may occur as a purely global phenomenon: in fact a graph of large girth, locally (around each vertex), looks just like a tree and is locally 2-colourable.

Graphs in which the local structure directly implies the chromatic number are perfect graphs. A *perfect graph* is a graph G such that for every induced subgraph of G, the size of the largest clique equals the chromatic number. Classes of graphs that are perfect include the empty graph, complete graphs, bipartite graphs, line graphs of bipartite graphs, and the graph complements of the latter two. Perfect graphs are also chordal graphs, comparability graphs and interval graphs which turn up in numerous applications. Important classes which are known to be perfect graphs are described in Hougardy (1998).

Grötschel et al. (1981) show that for perfect graphs the clique and the colouring problem can be solved in polynomial time. Unfortunately, this result makes use of the ellipsoid method in linear programming and its relevance is mainly theoretical while it has never been implemented in practice. Surprisingly, in spite of this result, a polynomial algorithm for recognising perfect graphs has been found recently by Cornuéjols et al. (2003). Its complexity is $\mathcal{O}(n^{10})$. At the current time, however, no implications of this result on the graph colouring problem have been published.

4.4. Benchmark instances and applications

The principal goal of our study is to empirically evaluate different approximate algorithms for graph colouring. We assume to have no *a priori* knowledge on a graph except the features that are easily retrieved, like size, density, and some basic structure. Consequently, we assume to know nothing about the chromatic number before solving an instance. This is the most general situation and the most likely to occur in practical cases.

The literature on methods for graph colouring has focused on a set of instances which is publically available. The advantage of this is the possibility of having immediate benchmarks with results attained by other existing algorithms thus making easier the comparison. We conform to this praxis and use the same set of instances for testing the algorithms developed by us. The need for using the same instances arises in our case to substantiate the results of the known algorithms that we re-implement and to make the results reliable. Nevertheless, this approach has also some drawbacks which were pointed out in Section 3.6 and therefore we will also test our algorithms on large classes of differently structured, random graphs constructed with the generator of Culberson et al. (1995). The analysis on these graphs will be presented in Section 4.13. The benchmark graphs are instead used more extensively throughout the chapter also for testing construction heuristics and iterative improvement algorithms and we introduce them here.

The benchmark instances are collected by the DIMACS Computational Challenge on "Graph Colouring and its Generalisations".[6] They consist of 125 graphs arising in different contexts. Since we saw that results may be different in graphs with different structure, we maintain them grouped in classes according to similar structure and study algorithms separately in each class. The presentation of these instances constitutes also an occasion to stress even more the wide field of practical applicability of graph colouring.

[6] M. Trick. "Computational Series: Graph Coloring and its Generalizations." August 2001. http://mat.gsia.cmu.edu/COLOR04/. (February 2005).

Random graphs: This class is based on $G_{n,p}$ graphs, which have n vertices and are generated by including each of the $n(n-1)/2$ possible edges independently at random with probability p. No *a priori* knowledge on the chromatic number or on lower bounds is given, although Johri and Matula (1982) provide some probabilistic indications. The instance identifiers are `DSJC`$n.p$, with $n \in \{125, 250, 500, 1000\}$ and $p \in \{0.1, 0.5, 0.9\}$, and have remained the same since their first use in the experimental study of Johnson et al. (1991).

Geometric random graphs: These graphs are generated as follows: first, n vertices are placed at random positions in a two dimensional unit square in the Euclidean plane; then, edges are added between any pair of vertices u and v whose Euclidean distance is smaller than a given value $d \in (0, \sqrt{2}]$. The graph denoted as `DSJR500.5` is such a graph with 500 vertices and $d = 0.5$, while the graph `DSJR500.1c` is the complement of such a graph with 500 vertices and $d = 0.1$. For a correspondence between d values and edge density see Figure 4.2 on page 96.

Queen graphs: The n-queens problem asks whether it is possible to place n queens on a $n \times n$ grid such that no pair of queens attacks each other. This problem can be posed as a graph colouring problem and a feasible solution to the n-queens problem exists if, and only if, the graph made of the vertices corresponding to the squares of the grid and joined by edges if the squares are on the same row, column, or diagonal, has a feasible colouring with n colours. A theorem shows that the $n \times n$ queens graph is n-colourable whenever $(n \bmod 6)$ is 1 or 5. This condition is necessary and sufficient for $n < 12$, however, it is not necessary for all n and counterexamples exist.[7] These graphs are highly structured instances and sparse, as their edge density decreases with increasing size.

Leighton graphs: Leighton graphs are random graphs with a predetermined chromatic number. The graphs are constructed by first grouping vertices into k sets, representing the colour classes, and then assigning edges only to vertices that belong to different sets. This guarantees an upper-bound to the chromatic number. The upper-bound is then made also a lower-bound by implanting cliques of sizes ranging from 2 to k into the graph. The vertices of the cliques are chosen in such a way that the chromatic number of the graph will not be larger than k. Additionally, Leighton graphs use a limit on the total number of edges and all graphs have a density lower or equal to 0.25. The graphs are denoted as `le450_`kx, where 450 is the number of vertices, k is the chromatic number of the graph and a letter $x \in \{a, b, c, d\}$ is used to distinguish different graphs with same characteristics but generated with different random seeds and with c and d having higher edge density than a and b.

Flat graphs: These graphs are random graphs generated consequently to an equi-partition of vertices in k independent sets. For a given density of the graph, edges are then distributed as equitably as possible between pairs of vertices belonging to different sets. For a discussion on how this is done we refer to Culberson et al. (1995). The number of sets k is an upper bound to the chromatic number of the graph. There are six instances, denoted as `flat`n_k, with $n = 300$ and $k \in \{20, 26, 28\}$ and with $n = 1000$ and $k \in \{50, 60, 76\}$.

[7] V. Chvátal. "Coloring the queen graphs." February 2004. `http://www.cs.concordia.ca/~chvatal/queengraphs.html`. (July 2005).

WAP graphs: These graphs arise in the design of transparent optical networks (Zymolka et al., 2003). In particular, the GCP is used to find an assignment of a minimal total number of wavelengths, given some pre-established light-path routing. The instances have the identifier `wap_0`m`a`, $m = \{1, \ldots, 8\}$. All instances have a clique of size 40. For some instances, the chromatic number is known.

Mycielski graphs: A Mycielski graph of order k is a triangle-free graph with chromatic number k. A triangle free graph has no cycles of length three and its clique number is two. These graphs are important because they show that an indefinitely large chromatic number is possible for graphs with a small clique number. They are denoted `myciel`$(k-1)$.

Insertions graphs: The insertions graphs are also triangle free graphs. They are obtained by a succession of graphs constructed by means of an extension of the Mycielski transformation. Shortly, given the insertion graph I_i^k the transformation produces a new graph I_{i+1}^k by introducing $n \cdot (k+1) + 1$ vertices, where n is the number of vertices of I_i^k, and edges in such a way to maintain the general properties of the graph. Graph I_1^k is the graph with two vertices and one edge. They were proposed by Caramia and Dell'Olmo (2002b) and are named as k`-insertions_`i.

Full Insertions graphs: Full Insertions graphs extend even further the Mycielski transformation. The mechanism is similar to the Insertions graphs where $n \cdot (k+1) + k + 2$ vertices are inserted at each transformation, however, they are not anymore triangle free graphs. We indicate them as k`-FullIns_`I, where k and I are two parameters which determine the transformation. The introduction of Insertions and Full Insertions graphs is quite recent and no theoretical proof concerning their chromatic number exists (Caramia and Dell'Olmo, 2002a).

Quasigroup Graphs: A Quasigroup is an algebraic structure on a set with a binary operator. The constraints on this structure define a Latin square, that is a square matrix with elements such that entries in each row and column are distinct. A Latin square of order n has n^2 vertices and $n^2(n-1)$ edges, corresponding to $2n$ cliques, a clique per row/column, each of size n; it is known that the chromatic number of Latin square (or Quasigroup) graphs is n. We denote Latin square graphs defined by Quasigroups as `qg.order`n. The instance `latin_square_10` stems from the design of Latin square experiments in statistical analysis (Lewandowski and Condon, 1996). The chromatic number of this latter graph is unknown.

Jacobian Estimation Graphs: These graphs stem from a matrix partitioning problem in the segmented columns approach to determine sparse Jacobian matrices (Hossain and Steihaug, 2002). The instance identifiers are `abb313GPIA`, `ash331GPIA`, `ash608GPIA`, `ash958GPIA` and `will199GPIA`.

Course Scheduling Graphs: These graphs arise in a course timetabling problem where courses cannot be scheduled in a same timeslot if they share students. In this problem, the vertices correspond to the courses, while an edge exists between a pair of vertices, if a student visits both courses. Two instances, `school` and `school-nsh`, for the problem of scheduling 385 courses have been proposed by Lewandowski and Condon (1996). An upper bound for the chromatic number of both graphs is 14.

Other Graphs: We also solved other graphs from the DIMACS repository, which, however, resulted to be easily solvable (details on what we mean by easily solvable are given in Section 4.7). These are:

- instances from register allocation for variables in real codes (Lewandowski and Condon, 1996);
- graphs from Donald Knuth's Stanford GraphBase: Book, Game, and Miles graphs (Knuth, 1993);
- graphs that are almost 3-colourable, but have a hard-to-find four clique embedded (Mizuno and Nishihara, 2002).

Given that these graphs are not anymore interesting for approximate algorithms we skip their detailed description.

In Appendix C.1.1 we give a detailed account of numerical statistics on all the individual instances of the DIMACS repository. Except where differently stated, all results in terms of computation times reported in this chapter refer to a PC with a 2 GHz AMD Athlon MP 2400+ Processor with 256 KB cache and 1 GB RAM. In order to make possible comparisons among different machines, a benchmark code is available at the DIMACS web site. We report the results produced by its execution on our machine in Appendix C.1.3.

4.5. Graph reduction

In the chromatic number problem, a graph G can be reduced to a graph G' if for any feasible colouring of G' a feasible colouring for G can be easily derived by some construction rules. Such a reduced graph G' may be obtained in a preprocessing stage. Three rules, proposed in Cheeseman et al. (1991), can be used for this scope.

Rule 1: Remove all vertices in G that have a degree less than the size of the largest known clique $\widehat{\omega}(G)$. Knowing that the degree of a vertex is less than $\widehat{\omega}(G)$ guarantees that at least one colour that is not used in the set of adjacent vertices can be assigned without breaking feasibility.

Rule 2: Remove a vertex $v \in V$ if there is another vertex $u \in V$, $v \neq u$ and $(u, v) \notin E$, that is connected to every vertex to which v is connected (subsumption). In this case, any colour that can be assigned to u can also be assigned to v.

Rule 3: Merge vertices that must have the same colour, that is, if vertices are fully connected to a clique of size $k-1$, then these vertices can be merged into a single vertex that is connected to all vertices to which the original vertices were connected, because they must have the same colour. This rule can be used only when trying to colour the graph with a number of colours $k = \widehat{\omega}(G) + 1$ and needs therefore k to be fixed and it is a particular case of Rule 2.

These three rules can be applied in any order and, typically, if one rule applies, it makes possible further reductions through the other rules. Hence, the preprocessing stage applies these rules iteratively until no vertex can be removed anymore. The rules are trivial to implement. Rule 1 is done in $\mathcal{O}(|V|)$ and a clique can be found by a construction heuristic. Rule 2 and 3 are, instead, more expensive and require $\mathcal{O}(|V|^3)$. The overall reduction time is, however, insignificant in practice.

4.6. Exact methods

In integer linear programming, the graph colouring problem on G can be formulated as a *set covering problem*, where the ground set is V, the available sets are the maximal (inclusion-wise) independent sets of G. Alternatively, it can be expressed as a *set packing problem* on the same ground set, where the sets are all stable sets of G. Unfortunately, the number of variables grows exponentially with the size of G and linear relaxations require column generation techniques for their solution (Mehrotra and Trick, 1996). The lower bound they give on the chromatic number may, however, be tight and effective branch and cut (and price) algorithms can be obtained thereby. In this way, Mehrotra and Trick (1996) show that it is possible to solve random graphs up to 70 vertices (these results can be updated to about 90 vertices on current machines, Schindl, 2003) and geometric graphs up to 250 vertices.

A much simpler way to colour exactly a graph without the need of dedicated software for solving linear programming is by branch and bound (see Section 2.3.3), where each vertex is selected and coloured with the lowest feasible colour. When no feasible colour is available a new colour is introduced. In case the introduction of a new colour makes the number of colours used higher than the number of colours in the best legal colouring seen so far, the colouring is pruned and the search continues after backtracking to some previously coloured vertex. A particular branch and bound algorithm may choose as vertex to colour next the one with the highest *saturation degree*, that is, the number of adjacent vertices differently coloured, breaking ties in favour of vertices that are adjacent to the most as yet uncoloured vertices. We call this branch and bound algorithm, which was actually proposed by Johnson et al. (1991), BB-GCP.

Another very famous and simple exact approach for graph colouring is the Brelaz's modification of Randall-Brown's exact colouring algorithm (Brélaz, 1979). The algorithm is a backtracking searching technique with forward checking (see Section 2.3.3 on page 17). We focus on the implementation of Mehrotra and Trick (1996), in which two basic improvements suggested by Brélaz (1979) and corrected by Peemöller (1983), are also included. In this method, first a large clique (the larger the better) is determined and its vertices are permanently coloured; then, the remaining vertices are considered sequentially selecting first those adjacent to the largest number of differently coloured vertices. We denote this algorithm, which is available on line, Ex-DSATUR.[8] The performance of Ex-DSATUR is comparable to that of the column generation approach described above and introduced by the same authors, therefore we can reasonably confine ourselves to include in our study only Ex-DSATUR.

In Figure 4.1, we compare BB-GCP and Ex-DSATUR on random graphs of size $\{40, 45, \ldots, 95, 100\}$ and density $p = \{0.1, 0.5, 0.9\}$. For each size-density pair, five graphs were generated with different random seeds and on each of these graphs the two algorithms were run with a time limit of 2 hours.

The behaviour of the two algorithms is similar for density 0.5 and 0.9. For density 0.1, Ex-DSATUR outperforms BB-GCP, which exhibits large variability. The reason for this result is due to the fact that the random graphs generated with size less or equal than 100 and low edge density have very likely $\omega(G)$ equal to $\chi(G)$ or $\chi(G) - 1$. A graph for which $\omega(G) = \chi(G)$ is called *1-perfect* and can be easily coloured, provided that a maximal clique can be found efficiently (Coudert, 1997). Finding a maximal clique

[8] M. Trick. "Network Resources for Coloring a Graph". October 1994. http://mat.gsia.cmu.edu/COLOR/color.html (February 2005).

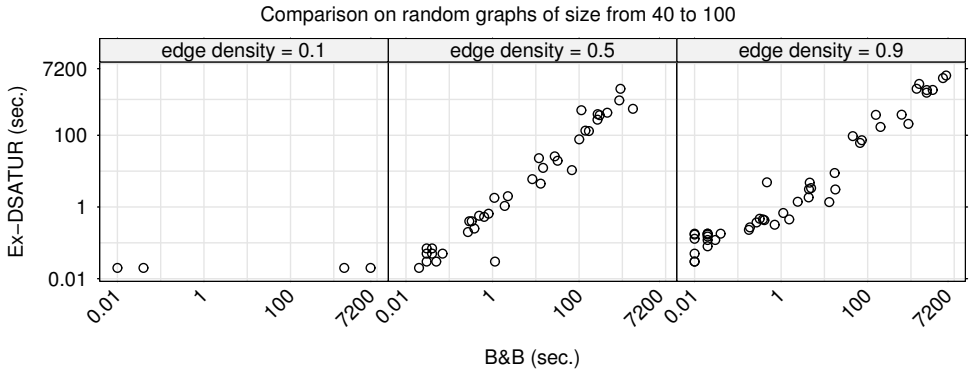


Figure 4.1.: The exact algorithms BB-GCP and Ex-DSATUR run on random graphs of size between 40 to 100. The size grows at increments of 5 and at each size 5 graphs are solved. In the graph, scales are logarithmic. At density 0.1 many graphs are omitted because not solved within the given time limit by BB-GCP. At density 0.5, graphs with large size (95 and 100) are omitted because they are not solved by both algorithms within 2 hours.

allows to fix the colours of its vertices and to abort the search as soon as a colouring of cardinality $\omega(G)$ is found. Similarly, with $\omega(G)$ close to $\chi(G)$ the pruning of the search tree can be very effective. BB-GCP has, instead, large and variable run time because many colourings are equally good and all must be explored. For graphs in which $\omega(G) < \chi(G)$, Ex-DSATUR does not outperform BB-GCP because the search must exhaustively enumerate all potential colourings that would improve on $\chi(G)$, which takes exponential time. A further observation is that graphs of edge density 0.5 result for both algorithms harder to solve than graphs of edge density 0.9.

In addition to this, we studied the two algorithms on a range of Uniform and Geometric random graphs generated by the algorithm of Culberson et al. (1995). In Figure 4.2 we show how the characteristics of such graphs vary in relation to the edge probability and the vertex distance in the plane. The characteristics represented are the average degree, the standard deviation, and the range of vertex degree. The graphs are all of size 30, and statistics are normalised over the size; when increasing instance size, the patterns remain equal. In general, we note that Geometric graphs exhibit higher variability of the vertex degree than Uniform random graphs. From the box-plots in the lower part of Figure 4.2, that represent the computation time of the two exact algorithms on 5 graphs per graph parameter, we observe that BB-GCP has strong problems with low density graphs while Ex-DSATUR does not appear to be influenced by any of the characteristics of these graphs (at least for size 30). For BB-GCP, the threshold of the vertex degree corresponding to graphs that are not easily solvable is shifted higher in Geometric graphs indicating that probably other characteristics of the graph influence the hardness with respect to this algorithm.

Other exact algorithms for graph colouring with performances similar to the algorithms described here exist in the literature. The best algorithm according to the results of the DIMACS benchmark suite presented at the DIMACS Computational Symposium held in 2002 is the multistage branch and bound algorithm by Caramia and Dell'Olmo (2002c). Apparently this algorithm can even solve some graphs of size 500 with computation times around 15 minutes. Integer programming formulations are presented and solved by Diaz and Zabala (2002) and Gomes and Shmoys (2002). This latter contribution is particularly interesting. Three different encodings of the problem are compared:

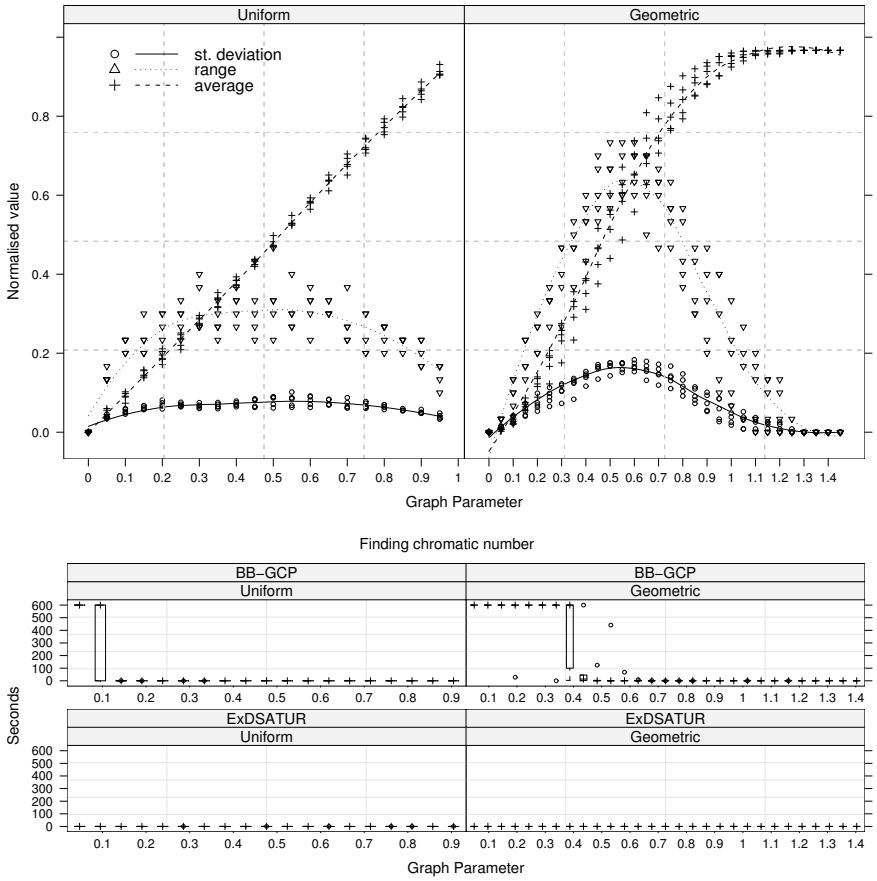


Figure 4.2.: Uniform and Geometric random graphs of size 30. The first plot (upper part) reports the statistics for 5 graphs generated in correspondence of each graph parameter: the edge probability for uniform graphs and vertex distance in the plane for geometric graphs. The second figure represents the computation time required to colour the graphs using one of the two exact algorithms introduced in this section. The range of graphs which are hard for BB-GCP appear larger on Geometric graphs. It is not clear, however, which other characteristic of the graphs, besides their density, makes the graphs harder to solve for BB-GCP.

a CSP formulation, an integer programming set packing formulation, and a SAT formulation; two exact solution approaches are used for solving the three formulations: a CSP based approach for the CSP and IP formulation, and a SAT approach for the SAT formulation. The comparison is accomplished only on the structured Quasigroup instances and no method is found to uniformly dominate the others. However, encoding GCP as a CSP problem or as a SAT problem can be non manageable for large graphs due to the amount of memory used to make explicit all the constraints. Furthermore, Hoos (1999b) shows that encoding CSP in SAT problem is not a very successful approach. Apparently, therefore, encoding problems into other problems which entail an increase in the number of variables is less appealing than using algorithms that directly exploit the domain knowledge of the problem itself.

Vasquez (2004) developed an exact algorithm for solving Queens graphs. He exploits the intrinsic structure of these graphs and obtains new state-of-the-art results. Herrmann and Hertz (2002) proposed another general approach based on the attempt to determine the smallest possible induced subgraph which has the same chromatic number as the original graph. This algorithm is particularly well suited when a k-colouring is known

Function greedy_algorithm($G(V, E), \pi$);
$k = |V|$;
$C_1 = \{v_{\pi(1)}\}, C_2 = \emptyset, \ldots, C_k = \emptyset$;
for $i = 2, \ldots, |V|$ **do**
  $l = \min\{k : \forall v_j \in C_k, (v_{\pi(i)}, v_j) \notin E\}$;
  $C_l = C_l \cup \{v_{\pi(i)}\}$;
end
Let $k = \max\{h : C_h \neq \emptyset\}$;
return the k-colouring, i.e. k and $C_1, \ldots, C_k$;

Algorithm 4.1: Greedy construction heuristic.

for a given subgraph and it has to be proved that no $(k-1)$-colouring exists.

This section allows us to conclude that in the general case looking for an exact answer to the GCP for graphs of size larger than 100 vertices may be extremely expensive in terms of computation time and the use of approximate solution methods becomes necessary.

4.7. Construction heuristics

Construction heuristics build feasible colourings in an incremental way. They start from an empty assignment and iteratively colour the vertices until all vertices are coloured.

Sequential heuristics are based on the *greedy heuristic*, that is, at each construction step a vertex is coloured with the lowest feasible colour. Given a set of empty colour classes $\{C_1, \ldots, C_k\}$, where $k = |V|$, and a permutation π of $\{1, \ldots, |V|\}$, the greedy algorithm assigns at iteration i the vertex $\pi(i)$ to the colour class with the lowest possible index such that the partial colouring obtained after colouring the vertex $\pi(i)$ remains feasible. This procedure is formalised in Algorithm 4.1.

Clearly, the result of the greedy heuristic depends on the permutation π. The *randomly ordered sequential heuristic (*ROS*)* uses a random permutation. Yet, better criteria can be used and the vertex order which determines π can be static, *i.e.*, computed once at the beginning, or dynamically, *i.e.*, recomputed for each subgraph $G_{i+1} = G_i \setminus v_i$ with v_i being the last vertex coloured.

Static sequential methods are dominated by dynamic ones as shown by Johri and Matula (1982). One of these dynamic orders is the DSATUR *heuristic* derived from the Ex-DSATUR algorithm of Brélaz (1979). Contrary to the exact method it does a unique, greedy construction and it ends when all vertices are coloured. No clique number is searched and no backtracking is performed. In DSATUR, the vertices are first arranged in decreasing order of degree and a vertex with maximal degree is inserted in C_1. Then, at each construction step, the next vertex to be inserted is chosen according to the saturation degree. The vertex with the maximal saturation degree is chosen and inserted into the colour class with the lowest possible index such that the partial colouring remains feasible. Ties are broken preferring vertices with the maximal number of adjacent, still uncoloured, vertices; if ties still remain, they are broken randomly.

The *recursive largest first (*RLF*)* heuristic of Leighton (1979) looks at the problem from the partitioning perspective. RLF iteratively constructs a colour class C by first assigning to it a vertex v with maximal degree from the set V' of still uncoloured vertices. Next,

Function RLF($G(V, E)$);
$k = |V|$;
$C_1 = \emptyset, C_2 = \emptyset, \ldots, C_k = \emptyset$;
$U = V, i = 1$; %U is the set of uncoloured vertices
while $U \neq \emptyset$ **do**
  $V' = U, U = \emptyset$;
  Let $v \in V'$ be a vertex for which $|A_{V'}(v)|$ is maximal;
  $C_i = C_i \cup \{v\}$;
  $U = A_{V'}(v), V' = V' \setminus A_{V'}(v)$; % U set of uncoloured vertices which cannot be placed in V
  while $V' \neq \emptyset$ **do**
    Let $v \in V'$ be a vertex for which $|A_U(v)|$ is maximal; % the edges in U are minimised
    $C_i = C_i \cup \{v\}$;
    $U = U \cup A_{V'}(v), V' = V' \setminus A_{V'}(v)$;
  end
  $i = i + 1$;
end
$k = i - 1$;
return the k-colouring, i.e. k and $C_1, \ldots, C_k$

Algorithm 4.2: Recursive Largest First heuristic

all the vertices in V' that are adjacent to v are removed from V' and inserted into a set U, which is initially empty; U is the set of vertices that cannot be added anymore to colour class C. Then, while V' is not empty, iteratively a vertex $v \in V'$ is chosen that has the maximal number of edges with vertices in U; v is added to C and all vertices in V' adjacent to v are moved to U. These steps are iterated until V' is empty and the same steps are repeated with the residual graph consisting only of vertices in U. Algorithm 4.2 shows the details of this procedure.

DSATUR and RLF are particularly popular heuristics and the best performing reported in the literature. Moreover, DSATUR and RLF are representative of the two different strategies for colouring a graph, as pointed out by (Culberson, 2001): the heuristics that colour a graph by iteratively selecting a vertex according to the number of colours already assigned to the adjacent vertices; and the heuristics that colour a graph by selecting large independent sets and using vertex degree information. De Werra (1990) analyses the theoretical properties of these sequential methods and shows that DSATUR and RLF are exact methods for bipartite graphs.

Experimental analysis. We compare the three construction heuristics DSATUR, RLF, and ROS. We do not expect ROS to be competitive, yet we include it in the analysis since it is the simplest method to construct a colouring, and hence the minimal requirement for any other algorithm is to do at least better than it. We run each heuristic 10 times on all benchmark instances introduced in Section 4.4. We are therefore in a *"several runs on various instances"* experimental design (see Section 3.6.6). The need for collecting replicates arises by the fact that in the three heuristics ties are at some point broken randomly and hence their result is stochastic. Numerical results are reported in Appendix C.1.1. Here, we show the analysis of the results by means of the rank-based Friedman test for all-pairwise comparisons (Equation 3.19 on page 62). The plots for the 95% simultaneous

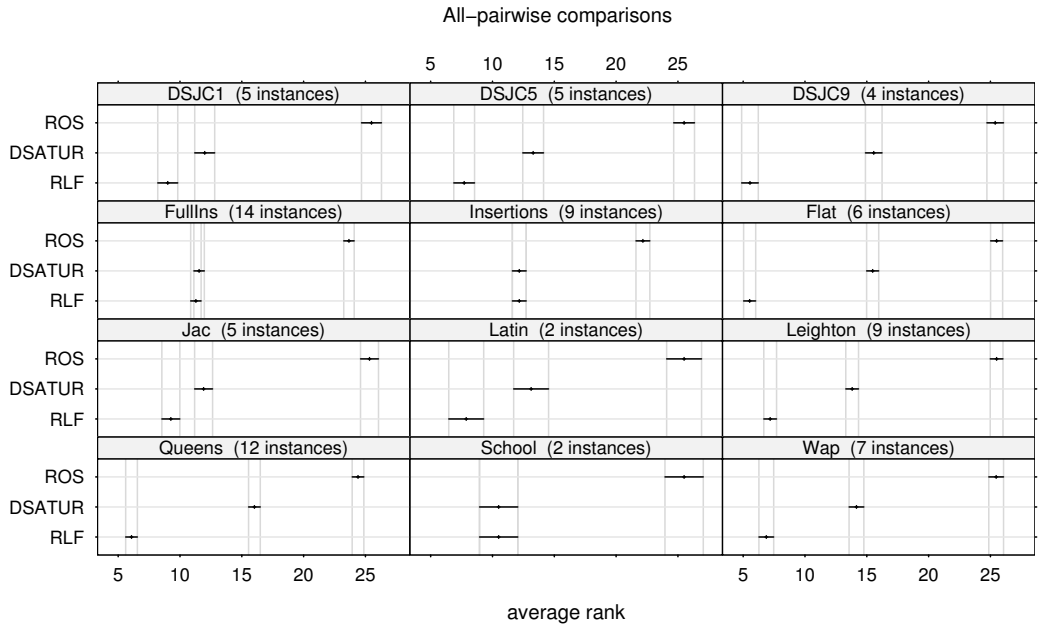


Figure 4.3.: Simultaneous confidence intervals for all-pairwise comparisons on construction heuristics. Each algorithm was run 10 times per instance.

confidence intervals of the all-pairwise comparisons are reported in Figure 4.3 (for a description on how to read this plot we refer back to page 53). With 3 algorithms and 10 runs per algorithm the average rank for a solution produced by one of the heuristics is a number in $[0, 30]$. We give preference to this test only because we deem easier to understand average ranks than the number of colours averaged over the instances. The other two methods, Tukey's HSD and permutation test, were also applicable in this case.

RLF clearly performs better than DSATUR in most instance classes, and both algorithms are always by a large margin better than ROS. RLF is not significantly better than DSATUR only on the Full Insertions, the Insertions, and the Course Scheduling (School) graphs. We remark that in the Latin square graphs and in the Course Scheduling only two graphs are available and statistical inference is less meaningful. The results on the Course Scheduling instances are however interesting for the understanding of the mechanism of the analysis procedure. The test based on ranks of Figure 4.3 tells indeed that differences are not significant. A similar test based on permutations (see Section 3.5), where results are not transformed into ranks, indicates, instead, that DSATUR is significantly better. Looking in-depth at the numerical results we have that DSATUR attains median results of 15 and 27 colours on the two instances, whereas RLF attains 25 and 25 colours. Clearly permutation tests weight more the gain 15-25 than 25-27 while rank tests loose this information. On the random graphs with edge density 0.1 we have an opposite effect: the permutation test procedure does not distinguish anymore differences between DSATUR and RLF while the Friedman test does. The same inference, holds, instead in all other cases.

From the point of view of the computation time, although RLF and DSATUR have the same time complexity $\mathcal{O}(|V|^3)$, RLF is in practice more costly. The complexity of the ROS heuristic is instead $\mathcal{O}(|V|^2)$. Klotz (2002) proposes a variant of RLF that works in $\mathcal{O}(|V|^2)$. After the first iteration, a vertex $u \in V$ to be added to class C_i is chosen to be the one with the highest number of vertices which are adjacent to both u itself and the last coloured

vertex v (*i.e.*, $\arg\max_{u \in V} |A(u) \cap A(v)|$). In practice, however, this method performs worse than DSATUR.

In the upper plot of Figure 4.4 we show the relation between the effective computation time and the size of an instance. The instances used are all instances from the DIMACS repository. For the largest instance with 10000 vertices, RLF takes a median time of about 18.5 seconds to colour the graph with 101 colours while DSATUR takes 3 seconds to colour it with 103 colours and the ROS heuristic less than 1 second using 106 colours. The superimposed lines are polynomial smoothed regression lines. The graph is in logarithmic scale, therefore, if the computation time were polynomial to the size of the instance, we would see that the points lay on a straight line. Evidently, while this could seem the case for ROS and DSATUR, it is not the case for RLF, that must be influenced by other features of the graph. The density of the graph and the final number of colours are possible candidates for having a significant impact.

The influence of these features is shown in the lower plot of Figure 4.4. Apparently, both edge density and colouring number help to explain the behaviour of RLF whose computation time grows with increasing density and increasing number of colours. ROS's and DSATUR's run time, instead, remain, at least on these instances unaffected by these features. The graphs for this second experiment were created using the generator by Culberson et al. (1995). They consisted in 370 graphs of size 1000 with different structural characteristics. The generator allows to hide in the graph an arbitrary colouring number which becomes an upper bound for the chromatic number of the generated graph. Clearly, the number of colours which is possible to hide in a graph is not totally independent from edge density and hence one of these two variables could be enough to explain high computation times for RLF, together with instance size.

In conclusion, the results here presented indicate that the RLF heuristic remains constantly the best in terms of quality maintaining reasonably low computation times. For these reasons, we will adopt it for generating the initial solution to all SLS algorithms that we will study in Section 4.10. Culberson et al. (1995) pointed out that there are cases in which the initial colouring can negatively bias the final performance of a local improvement algorithm, independently from its quality. Nevertheless, we were unable to find a confirmation to this conjecture in our study.

Benchmark instances which are easy. We use the RLF construction heuristic and the graph reduction rules of Section 4.5 to distinguish between easy and hard instances in the DIMACS instance set. More specifically, we consider a graph to be *easy* if at least one of the following conditions occurs:

Condition 1: The reduction by means of Rule 1 and Rule 2 renders the graph a null graph, that is, a graph without vertices and edges.

Condition 2: The RLF heuristic produces an upper bound that is equal to the known chromatic number.

Condition 3: The lower bound $\widehat{\omega}_G$ for ω_G attained heuristically and the upper bound for χ_G produced by RLF coincide, that is, the solution returned by RLF is optimal.

Condition 4: One of the conditions 2 or 3 is satisfied after the graph reduction.

Based on this definition, 45 of the 125 instances of the DIMACS repository are classified as easy. The remaining 80 instances are classified as hard and will be object of the experiments in Section 4.11. Few instances, 18 easy and 3 hard ones, are also found to

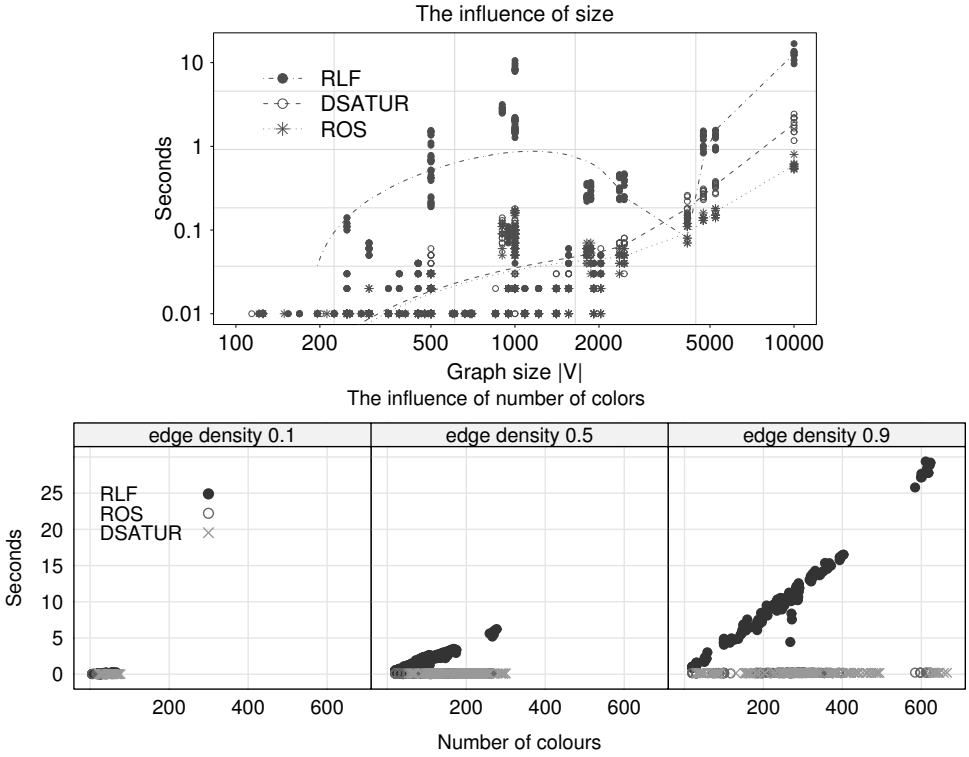


Figure 4.4.: On the upper part, the scatter plot shows the relation between computation time and graph size for the three construction heuristics. A point corresponds to the median time reported on 10 runs on the instances of the DIMACS repository. Axes are in logarithmic scales and the superimposed lines are smoothed polynomial regression lines. Size alone does not explain the behaviour of RLF. On the lower part, scatter plots for the computation time of the heuristics and the number of colours found on instances of three different edge densities. A point corresponds to one single run on one of the 370 randomly generated instances of size 1000.

be disconnected graphs. Statistics on all the instances are given in Table C.1 of Appendix C.1.1.

For determining the clique number $\widehat{\omega}_G$ in Condition 3 we used two heuristics that are among the state-of-the-art algorithms for solving the clique problem: a variant of the "semi-exhaustive greedy" scheme for finding large independent sets described by Johnson et al. (1991)[9] and the reactive tabu search from Battiti and Protasi (2001)[10]. We took the best outcome of the two.

Among the hard instances, graph reduction is effective only for the WAP and Full Insertions instances, while many other graphs are unaffected. Intuitively, algorithms produce better results in terms of quality and time when run on reduced graphs. This is confirmed experimentally also for the algorithms of our study. However, given that the reduceable graphs are few and that we aim at making our results comparable to those of the literature, where graph reduction was never used, we decided not to apply this preprocessing step before solving the hard graphs.

[9] Available through FTP at `ftp://dimacs.rutgers.edu/pub/dsj/clique`. (February 2005)
[10] A. Villani. "InterTools – Interactive Tools through the Internet: Max Clique". 1996. `http://rtm.science.unitn.it/intertools/clique`. (February 2005)

4.8. Iterative Improvement for graph colouring

There are three main approaches with corresponding different local search methods for solving the chromatic number problem: 1) considering a sequence of k-colouring problems with decreasing k, 2) operating on complete colourings with the number of colours free to vary, and 3) extending partial k-colourings. We will mainly focus on the first approach. For the second approach we will consider only one algorithm while for the third approach we will consider a semi-exhaustive method. Local search methods are possible in all the three cases.

Solving a sequence of k-colouring problems. An initial k_I is determined by a construction heuristics. For each $k < k_I$ a k-colouring problem is then solved with local search methods by formalising it as an optimisation problem. The set of candidate solutions $\mathcal{S}$ comprises all (complete) colourings $\mathcal{C}$ of the graph G (feasible and infeasible). The set $\mathcal{S}$ is therefore defined as the Cartesian product $\mathcal{S} = \Gamma \times \Gamma \ldots \times \Gamma = \Gamma^{|V|}$, and $|\mathcal{S}| = k^{|V|}$. In fact, two colourings that differ only for a permutation of the labels associated with the colours are to be considered equal and the number of distinct complete colourings could be reduced to $k^{|V|}/k!$. However, in practice, the reduction of $\mathcal{S}$ does not take place since there is no efficient method to implement it.

The evaluation function is usually defined as the number of edges that create conflicts in a colouring:

$$f(\mathcal{C}) = \sum_{i \in \Gamma} |E_{C_i}(C_i)|, \mathcal{C} \in \mathcal{S}.$$

The goal is to find a solution $\mathcal{C}^*$ such that $f(\mathcal{C}^*) = 0$.

An alternative choice for the evaluation function would be the number of vertices involved in a conflict: $f'(\mathcal{C}) = |V^c|$. The two evaluation functions are not in a one-to-one correspondence. Indeed, because of Remark 4.2, it holds that

$$f(\mathcal{C}) = \sum\nolimits_{i \in \Gamma} |E_{C_i}(C_i)| = \frac{1}{2} \sum_{i \in \Gamma} \sum_{v \in C_i} |A_{C_i}(v)|$$

and we can derive that, in general,

$$f' = |V^c| = \sum_{i \in \Gamma} |\bigcup_{v \in C_i} A_{C_i}(v)| \neq \frac{1}{2} \sum_{i \in \Gamma} \sum_{v \in C_i} |A_{C_i}(v)| = \sum_{i \in \Gamma} |E_{C_i}(C_i)| = f.$$

Hence, the two evaluation functions could yield different performance in the local search. In the literature however the first evaluation function is the mostly used and we focus on it.

When a feasible colouring is found, the next decision problem is defined by reducing the number of colours available by one and reapplying local search. This is done until for a certain number of colours k a feasible colouring cannot be found. Then, the best feasible colouring identified uses $k + 1$ colours, and the outcome of the search is $\widehat{\chi}_G = k + 1$.

Varying the number of used colours. In this case the number of colours used may increase and decrease at run time. The number of colours returned, $\widehat{\chi}_G$, is the lowest number of colours for which a feasible colouring was found during the search. An evaluation function that leads the search towards colourings with fewer colours and towards

feasible colourings was introduced by Johnson et al. (1991) and is given by:

$$f(\mathcal{C}) = -\sum_{i=1}^{K} |C_i|^2 + \sum_{i=1}^{K} 2|C_i||E^c_{C_i}|, \quad \forall \mathcal{C} \in \mathcal{S}. \tag{4.1}$$

It can be proved that local optima with respect to this function correspond to feasible colourings. Nevertheless, the search can be biased towards unbalanced colour classes and the optimal solution does not necessarily use the minimal number of colours. However, these drawbacks appear to have little importance in practice (Johnson et al., 1991).

Extending partial *k*-colourings. In this case the number of colour classes is again fixed to a target value k, and a solution $\mathcal{C} = \{C_1, C_2, \ldots, C_k, C_I\}$ is a partial feasible colouring together with a single improper class C_I containing all vertices that are not part of the partial feasible colouring. The goal is extending the partial colouring to the vertices in C_I. The evaluation function could be $f(\mathcal{C}) = |C_I|$, which has to be minimised to zero. Better performances were, however, shown by (Morgenstern and Shapiro, 1990) with

$$f(\mathcal{C}) = \sum\nolimits_{v \in C_I} \left(1 + \frac{d(v)}{d(G)}\right)$$

in which vertices are weighted according to their degree. This evaluation function does not bias the search towards skewed distribution of classes.

4.8.1. Neighbourhood structure

We introduce six neighbourhood structures for local search on the GCP. Six of them find application in solving the k-colouring problem, while the last one is applied when k is left variable. While the first and the last neighbourhood structures presented are well known in the literature, all the others are original contributions of this thesis.

Definition 4.1 (one-exchange) The neighbourhood $\mathcal{N}_1(\mathcal{C})$ is the set of colourings $\mathcal{C}'$ obtained from $\mathcal{C}$ by changing the colour of exactly one vertex that is involved in a conflict. Formally,

$$\begin{aligned} \mathcal{N}_1(\mathcal{C}) \quad = \quad & \{\mathcal{C}' : \exists i \in \Gamma \text{ and } \exists v \in V^c : C'_{\varphi(v)} = C_{\varphi(v)} \setminus \{v\} \text{ and } C'_i = C_i \cup \{v\} \text{ and} \\ & C'_l = C_l \; \forall l \in \Gamma \setminus \{i, \varphi(v)\}\} \end{aligned}$$

Definition 4.2 (Swap) The neighbourhood $\mathcal{N}_2(s)$ is the set of colourings $\mathcal{C}'$ obtained from $\mathcal{C}$ by swapping the colour of a vertex involved in a conflict with the colour of another vertex. Formally,

$$\begin{aligned} \mathcal{N}_2(\mathcal{C}) \quad = \quad & \{\mathcal{C}' : \exists v \in V^c \text{ and } u \in V : C'_{\varphi(v)} = C_{\varphi(v)} \setminus \{v\} \cup \{u\} \\ & \text{and } C'_{\varphi(u)} = C_{\varphi(u)} \setminus \{u\} \cup \{v\} \\ & \text{and } C'_l = C_l \; \forall l \in \Gamma \setminus \{\varphi(v), \varphi(u)\}\} \end{aligned}$$

Definition 4.3 (Pair swap) The neighbourhood $\mathcal{N}_3(\mathcal{C})$ is the set of colourings $\mathcal{C}'$ obtained from $\mathcal{C}$ by swapping the colour of a pair of adjacent vertices, which belong to a same

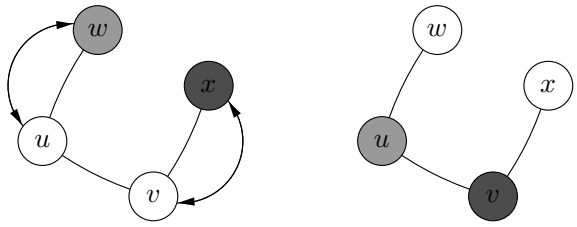


Figure 4.5.: A pair swap exchange. The colour of two vertices, currently in conflict, is exchanged with the colours of two adjacent vertices.

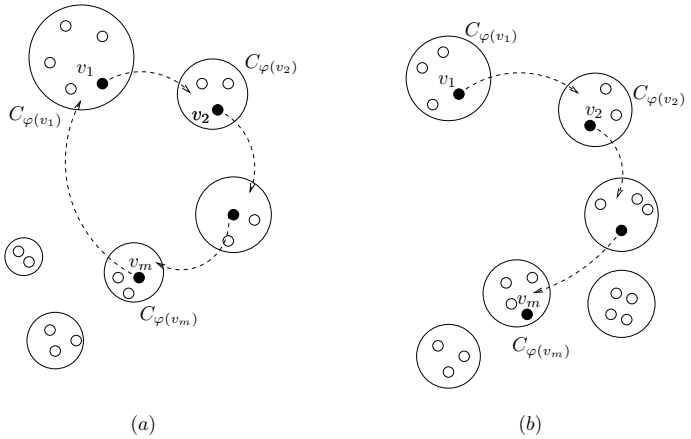


Figure 4.6.: The black vertices are the vertices involved in the cyclic and path exchange. (a) Cyclic exchange. (b) Path exchange.

colour class, with the colours of two other vertices that are each adjacent to one of the vertices of the pair and belong to mutually different colour classes. Formally,

$$\begin{aligned}\mathcal{N}_3(\mathcal{C}) \quad = \quad & \{\mathcal{C}' : \exists u, v \in V^c \text{ and } \exists w, x : w \in A_V(u), x \in A_V(v), \varphi(w) \neq \varphi(x) : \\ & C'_{\varphi(u)} = C_{\varphi(u)} \setminus \{u, v\} \cup \{w, x\}, \\ & C'_{\varphi(w)} = C_{\varphi(w)} \setminus \{w\} \cup \{u\}, \\ & C'_{\varphi(x)} = C_{\varphi(x)} \setminus \{x\} \cup \{v\}, \\ & C'_l = C_l, \forall l \in \Gamma \setminus \{\varphi(u), \varphi(w), \varphi(x)\}\}\end{aligned}$$

An example of pair swap is given in Figure 4.5.

The next two neighbourhoods we introduce are the very large scale neighbourhoods (VLSN) based on cyclic and path exchanges. In Figure 4.6, we represent a colouring as vertices grouped into colour classes.

A *cyclic exchange* of length m is a sequence of vertices $(v_1, \ldots, v_m)$ that moves vertex v_i from subset $C_{\varphi(v_i)}$ to subset $C_{\varphi(v_{i+1})}$ and v_m to subset $C_{\varphi(v_1)}$. A cyclic exchange does not change the cardinality of the subsets involved. In a path exchange, instead, the sequence of exchanges is not closed and the colour classes of the first and the last vertex in the sequence modify their cardinality. A *path exchange* can be seen as a generalisation of a one-exchange while a cyclic exchange can be seen as a generalisation of a swap.

Definition 4.4 (Cyclic exchange) The neighbourhood $\mathcal{N}_4(\mathcal{C})$ is the set of colourings $\mathcal{C}'$

obtained from $\mathcal{C}$ by *a cyclic exchange* of vertices in the colour classes. Formally,

$$\begin{aligned}\mathcal{N}_4(\mathcal{C}) \;=\; & \{\mathcal{C}' : \exists\, V' = v_1, v_2, \dots v_m \subseteq V, \varphi(v_i) \neq \varphi(v_j), \forall i, j \in \{1, \cdots, m\} : \\ & C'_{\varphi(v_{i+1})} = C_{\varphi(v_{i+1})} \setminus \{v_{i+1}\} \cup \{v_i\}, \forall i \in \{1, \cdots, m-1\}, \\ & C'_{\varphi(v_1)} = C_{\varphi(v_1)} \setminus \{v_1\} \cup \{v_m\}, \\ & C'_l = C_l, \forall l \in \Gamma \setminus \{\varphi(v_1), \cdots, \varphi(v_m)\}\}\end{aligned}$$

Definition 4.5 (Path Exchange) The neighbourhood $\mathcal{N}_5(\mathcal{C})$ is the set of colourings $\mathcal{C}'$ obtained from $\mathcal{C}$ by *a path exchange* of vertices in the colour classes. Formally,

$$\begin{aligned}\mathcal{N}_5(\mathcal{C}) \;=\; & \{\mathcal{C}' : \exists V' = v_1, v_2, \dots v_m \subseteq V, \varphi(v_i) \neq \varphi(v_j), \forall i, j \in \{1, \cdots, m\} : \\ & C'_{\varphi(v_{i+1})} = C_{\varphi(v_{i+1})} \setminus \{v_{i+1}\} \cup \{v_i\}, \forall i \in \{1, \cdots, m-2\}, \\ & C'_{\varphi(v_m)} = C_{\varphi(v_m)} \cup \{v_{m-1}\} \\ & C'_{\varphi(v_1)} = C_{\varphi(v_1)} \setminus \{v_1\} \\ & C'_l = C_l, \forall l \in \Gamma \setminus \{\varphi(v_1), \cdots, \varphi(v_m)\}\}\end{aligned}$$

It is important to note that the cyclic and path exchanges are not limited in length (except for being shorter than k). In this aspect, our VLSN is different from other two similar methods that work on large neighbourhoods: variable depth methods and ejection chains (see also 2.4.2). To the best of our knowledge, no variable depth search method has been devised for the graph colouring problem, while ejection chains were applied by González-Velarde and Laguna (2002), but the length of the step was limited to 3 successive one-exchanges.

Avanthay et al. (2003) analyse several other fancy neighbourhood structures that work with the k-colouring approach. However, none entails significant improvements over one-exchange neighbourhood when used in SLS algorithms and we avoid to consider them here.

Finally, we introduce a neighbourhood that may be used when dealing with feasible colourings and varying the number of colours, as introduced on page 102.

Definition 4.6 (Kempe exchange) The neighbourhood $\mathcal{N}_6(\mathcal{C})$ is the set of colourings $\mathcal{C}'$ obtained from $\mathcal{C}$ by a *Kempe chain exchange*. A *Kempe chain* is a set of vertices that form a maximal connected graph in the subgraph G' of G induced by the vertices that belong to two (disjoint) sets C_i and C_j of the partition $\mathcal{C}$. We recall that a connected graph is a graph where any two vertices are connected by a path. A Kempe chain *exchange* applied to a feasible colouring produces a new feasible colouring (a *Kempe chain neighbour*) by changing the colour of the vertices that belong to a specified Kempe chain K. An example of a Kempe chain is given in Figure 4.7. Formally, the neighbourhood is defined as:

$$\begin{aligned}\mathcal{N}_6(\mathcal{C}) \;= \{\mathcal{C}' : \exists\, K \subseteq (C_i \cup C_j), i \neq j : \;& C'_i = (C_i \setminus K) \cup (C_j \cap K), \\ & C'_j = (C_j \setminus K) \cup (C_i \cap K), \\ & C'_l = C_l, \forall l \in \Gamma \setminus \{i, j\}\}\end{aligned}$$

Note that feasibility is maintained and, therefore, the application of a Kempe chain is only reasonable for solving the GCP by varying the number of colours. In this case, the evaluation function given by Equation 4.1 simplifies to the first term.

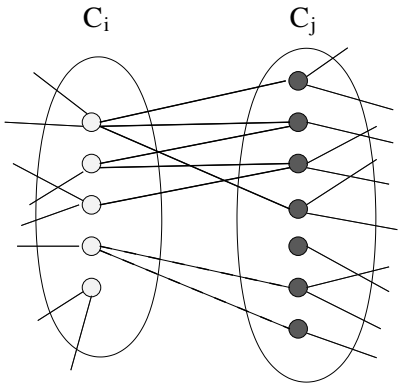


Figure 4.7.: Two Kempe chains (maximal connected components) are available between colour classes C_i and C_j and are indicated by the thick and dashed line. In a Kempe chain interchange the vertices belonging to a Kempe chain are swapped between the two colour classes.

Neighbourhood structures for graph colouring that may be used in the partial colourings approach have been studied by Morgenstern and Shapiro (1990). They introduce the *i-swap* which consists in removing some vertex v from the improper class C_I (that they call *impasse class*), adding it into a proper colour class C_i, and moving all vertices adjacent to v in C_i to C_I. They also generalise Kempe chains to *s-chains*, which expand the size of the neighbourhood such that more moves are made available when the quality of a solution is good. Successively, Morgenstern (1996) proposes a method that alternates i-swaps with s-chains and runs on a distributed computational environment. Finally, Blöchliger and Zufferey (2003) propose the use of *i-swaps* within a Tabu Search algorithm. These results are interesting and indicate that, similar peak performance may be attained by such an approach working on partial colourings.

4.8.2. Neighbourhood examination

Many of the SLS algorithms that we analyse use a best improvement search strategy where ties are broken randomly. The evaluation of a solution is done through incremental updates to increase the speed of the neighbourhood examination. Next, we describe how the examination is done in each neighbourhood.

One-exchange neighbourhood. Given a colouring $\mathcal{C}$ that uses k colours, the size of the neighbourhood $\mathcal{N}_1$ is $(k-1)\cdot|V^c|$. Since it depends on the number of vertices in conflict, the size is variable during the search. If vertex v moves from C_i into C_j, the evaluation function after an exchange can be computed as $f(\mathcal{C}') = f(\mathcal{C}) - |A_{C_i}(v)| + |A_{C_j}(v)|$. If we keep as additional data structure a matrix $\mathbf{\Delta}$ of $|V| \times k$ elements, where we record for all i, v the values $|A_{C_i}(v)|$, the evaluation of each neighbour can be done in $\mathcal{O}(1)$. The initialisation of the matrix is done once at the beginning in $\mathcal{O}(|V|^2)$. At each iteration, the matrix is updated and the entries that need to be considered are only those cells corresponding to the vertex that moved into a different colour class and the vertices adjacent to it. Hence, the worst case complexity for updating $\mathbf{\Delta}$ is $\mathcal{O}(|V|)$, but in practice it is much lower. In previous applications of tabu search (Fleurent and Ferland, 1996), the update of a similar data structure required $O(k|V|)$. In Algorithm 4.3 we give the algorithmic details of the initialisation and the update of $\mathbf{\Delta}$.

```
Function Initialise_Δ(G,φ);
Δ = 0;
for each v in V do
    for each u in A_V(v) do
        Δ(v, φ(v)) = Δ(v, φ(v)) + 1;
    end
end

Function Update_Δ(G,φ,v,C_i)  %C_i is the old colour class of v
for each u in A_V(v) do
    Δ(u, φ(v)) = Δ(u, φ(v)) + 1;
    Δ(u, c_old) = Δ(u, c_old) − 1;
end
```

Algorithm 4.3: Pseudo-code for the initialisation and update of the matrix $\mathbf{\Delta}$.

Swap neighbourhood. Every swap can be seen as a composition of two one-exchange (with the opportune adjustments, see Appendix A), and, the data matrix $\mathbf{\Delta}$ can be used for the evaluation of neighbours. Examining all possible swaps of neighbourhood $\mathcal{N}_2$ has a worst case complexity of $\mathcal{O}(|V^c||V|)$ and the size of the neighbourhood is variable depending on $|V^c|$.

Pair swap neighbourhood. Also in this case the evaluation of a neighbour can be obtained by composition of one-exchange moves, and, hence, using again the matrix $\mathbf{\Delta}$. The complete exploration of neighbourhood $\mathcal{N}_3$ entails a worst case complexity of $\mathcal{O}(|V|^4)$. In practice, keeping a list of conflicting edges E^c the complexity can be reduced to $\mathcal{O}(|E^c| \times |V|^2)$.

Cyclic and path exchange neighbourhood. The cyclic and path exchange neighbourhoods have exponential size. Ahuja et al. (2001b) showed that the problem of finding the best neighbour within a cyclic exchange neighbourhood can be modelled as the problem of finding the minimal cost cycle that uses at most one vertex from each subset in a new directed graph $G' = (V', D')$, called the *improvement graph*, induced by the graph $G = (V, E)$ and the current colour partition $\mathcal{C} = \{C_1, \ldots, C_k\}$. The set of vertices of G' is $V' = \{1, \ldots, n\}$, each vertex in V' corresponding to exactly one vertex $v_i \in V$. The improvement graph contains the arc (i, j) if $v_j \in C_{\varphi(v_j)}$ in $\mathcal{C}$ and $\varphi(v_i) \neq \varphi(v_j)$, that is, if v_i and v_j belong to different colour classes. The set of vertices V' is split into k subsets $T_1, \ldots, T_k$, induced by $\mathcal{C}$; the elements of T_h are in one-to-one correspondence with the elements of C_h, $\forall h = 1, \ldots, k$. The cost associated to an arc (i, j) with $i \in T_{\varphi(v_i)}$ and $j \in T_{\varphi(v_j)}$ is the cost of inserting vertex i into and removing vertex j from $T_{\varphi(v_j)}$. It can be easily computed using the same data matrix $\mathbf{\Delta}$ defined for $\mathcal{N}_1 : c_{i,j} = |A_{C_{\varphi(v_j)}}(v_i)| - |A_{C_{\varphi(v_j)}}(v_j)|$.

Thompson and Orlin (1989) showed that there is a one-to-one correspondence between the cyclic exchanges in G, with respect to $\mathcal{C}$, and the subset-disjoint cycles in the improvement graph G'. Clearly, an improving cyclic exchange will correspond to a subset disjoint cycle of negative cost. Finding such a cycle corresponds to solving an *all-pairs shortest path* problem (Ahuja et al., 2001b), which can be done by an exact algorithm based on dynamic programming ideas. The complexity of the algorithm is $\mathcal{O}(n^2 2^k |D'|)$, where $|D'|$ is the number of arcs of the improvement graph, as described in Ahuja et al. (2001b). The dynamic program is therefore unlikely to be applicable to large graphs. Nevertheless,

some heuristic criteria can be adopted to prune the search and make the algorithm more efficient. We discuss them in Section 4.9.3 together with the difficulties that arise, instead, in using the same approach for finding also path exchanges.

After each cyclic or path exchange the update of the auxiliary data matrix $\mathbf{\Delta}$ is done by considering the exchanges as a composition of one-exchanges. Hence, the updating procedure has a worst case complexity of $\mathcal{O}(k|V|))$, when the exchange involves one vertex from each colour.

Kempe chain neighbourhood. Finding the best Kempe chain entails the examination of all possible $\binom{k}{2}$ pairs of colour classes in the k-colouring. Once a pair is identified, all connected components are considered, that are subsets of the union of the two colour classes. Since each class is proper, in each connected component vertices belonging to the two colour classes are interleaved. In practice, the search proceeds by considering for each vertex in one colour class all adjacent vertices in the other class that are not already included in the current connected component. The cost of this search is high and leads to conjecture that Kempe chains should work better on dense graphs where the colour classes tend to be small (Johnson et al., 1991). To speed up this examination, some auxiliary data structure can be kept in order to avoid the examination of a pair of colour classes C_i and C_j if they were discovered to contain a full Kempe chain, *i.e.*, $(K = C_i \cup C_j)$ in some previous steps; any full Kempe chain can be discarded, as it corresponds to simply changing the index of two colour classes and does not have any effect. However, this speed-up does not reduce the worst case complexity of the neighbourhood examination and, in practice, $\mathcal{N}_6$ is only effective with simulated annealing, which does not require an exhaustive examination of the neighbourhood. We will use Kempe chains only in the approach with k variable. In this case, given that each neighboring colouring is a feasible (proper) colouring, the evaluation function of Equation 4.1 for a neighbour $\mathcal{C}' \in \mathcal{N}_6$ is computed by $f(\mathcal{C}') = f(\mathcal{C}) - |C_i|^2 - |C_j|^2 + |C'_i|^2 + |C'_j|^2$, which can be done in constant time, provided that the number of vertices moving from one class to the other in the Kempe chain is memorised during the construction of the chain itself.

4.9. Analysis of neighbourhood structures for local search

In this section, we investigate the behaviour of the neighbourhood structures described when used inside an Iterative Improvement algorithm with best improvement strategy. We restrict the attention to the k-colouring problem.

4.9.1. Analytical results

In the Appendix A we report the details of an analytical study aimed at understanding whether the use of cyclic and path exchanges is profitable. In particular, we show that there exist configurations in the search space that are not solvable by one-exchanges but that are solvable by a cyclic or a path exchange. More specifically, we show that every Iterative Improvement in which the neighbourhood $\mathcal{N}_1$ is enlarged with the union of neighbourhoods $\mathcal{N}_2$, $\mathcal{N}_3$, $\mathcal{N}_4$ or $\mathcal{N}_5$ yields better quality performance than an Iterative Improvement based solely on $\mathcal{N}_1$. The best combination of neighbourhoods is the union

of cyclic and path exchanges $\mathcal{N}_4 \cup \mathcal{N}_5$, as it allows to solve all configurations solved by other reasonable combinations of neighbourhoods and some more that would not be solved otherwise (note that path exchanges include also one-exchanges).

Nevertheless, some issues remained open. We were, for instance, unable to show whether the combination of path and cyclic exchanges also dominates any combination with pair swap neighbourhood. Moreover, the analytical results do not provide any quantitative insight on which of the cyclic and path exchanges are more profitable for the search. In this section, we try to resolve these issues by an empirical analysis and we investigate the use of possible pruning rules for an effective exploration of the neighbourhood $\mathcal{N}_4 \cup \mathcal{N}_5$.

4.9.2. Computational analysis on small size graphs

We denote each Iterative Improvement algorithm by $\mathcal{II}$ and indicate in the index the neighbourhood that characterises it. We study the following neighbourhoods: $\mathcal{N}_1$, $\mathcal{N}_1 \cup \mathcal{N}_2$, $\mathcal{N}_1 \cup \mathcal{N}_2 \cup \mathcal{N}_3$, $\mathcal{N}_1 \cup \mathcal{N}_4$, $\mathcal{N}_2 \cup \mathcal{N}_5$, $\mathcal{N}_4 \cup \mathcal{N}_5$, and $\mathcal{N}_3 \cup \mathcal{N}_4 \cup \mathcal{N}_5$, which are also addressed in the Appendix A. Here we examine whether more complex neighbourhoods guarantee an increase in the capability of solving the problem independently from their computational cost. To this end, we first test the algorithms on very small graphs.

The very small graphs are generated as follows. For a given set of vertices V, with $|V| = n$, we generate the set $\mathcal{G}(n)$ of all connected non-isomorphic graphs on V. Then, for each graph $G \in \mathcal{G}(n)$, we solve the k-colouring problem with k fixed to $\chi(G)$. All possible *distinct* (or inequivalent) candidate colourings $\widehat{\mathcal{S}}_G$ of each graph $G \in \mathcal{G}(n)$, feasible and infeasible, are used as an initial solution for an Iterative Improvement algorithm.

In Table 4.2, we report for each n the number of graphs having chromatic number k, *i.e.*, the size of $\mathcal{G}(n,k) = \{G \in \mathcal{G}(n) : \chi(G) = k\}$. We then denote with $\mathcal{G}(n)$ the set union of these graphs through k (the size of this set gives rise to a sequence of numbers of connected non-isomorphic graphs which can be found on the "On-Line Encyclopedia of Integer Sequences"[11] and on the links therefrom). The set of non-isomorphic connected graphs were generated with the publically available program geng.[12] The number of distinct colourings of a graph $G(n,k) \in \mathcal{G}(n,k)$ using l colours corresponds to the number of partitions of an n-set into l non empty parts and is given by the Stirling's number of second kind $S(n,l)$ (Graham et al., 1994). In parenthesis we report the number of distinct colourings that use $2 \leq l \leq k$ colours, *i.e.*, $\widehat{\mathcal{S}}_{G(n,k)} = \sum_{j=2}^{k} S(n,j)$. The total number of distinct colourings is then obtain as

$$|\widehat{\mathcal{S}}_{\mathcal{G}(n)}| = \sum_{k=1}^{K} |\mathcal{G}(n,k)| \cdot |\widehat{\mathcal{S}}_{G(n,k)}|$$

where K is the largest chromatic number for graphs of that size. Already for graphs of $n = 7, 8, 9$ the size of $\widehat{\mathcal{S}}_{\mathcal{G}}(n)$ becomes very large and, therefore, we consider a sample of size 40.000 initial colourings from $|\widehat{\mathcal{S}}_{\mathcal{G}}(n)|$.

Let $\widehat{\mathcal{S}}^{\mathcal{II}}(G)$ be the set of all distinct colourings that are infeasible colourings and are local optima for an algorithm $\mathcal{II}$ on a graph G. This set can be attained by running $\mathcal{II}$

[11] N.J.A. Sloane. "The On-Line Encyclopedia of Integer Sequences." June 2005. http://www.research.att.com/~njas/sequences/ (June 2005).

[12] Brendan McKay. 1984-2004. http://cs.anu.edu.au/~bdm/nauty/ (December 2003).

$\chi(G)$	$n=3$	$n=4$	$n=5$	$n=6$	$n=7$	$n=8$	$n=9$
2	1(3)	3(7)	5(15)	17(31)	44(63)	182(127)	730(255)
3	1(4)	2(13)	12(40)	64(121)	475(364)	5036(1093)	80947(3280)
4		1(14)	3(50)	26(186)	282(714)	5009(2794)	149551(11050)
5			1(51)	4(201)	46(854)	809(3844)	27794(18001)
6				1(202)	5(875)	74(4110)	1940(20647)
7					1(876)	6(4138)	110(21109)
8						1(4139)	7(21145)
9							1(21146)
$\lvert\mathcal{G}(n)\rvert$	2	6	21	112	853	11117	261080
$\lvert\widehat{\mathcal{S}}_{\mathcal{G}(n)}\rvert$	7	63	756	14113	421555	22965511	2461096985
Sample	7	63	756	14113	40000	40000	40000

Table 4.2.: For different chromatic numbers (rows) the number the table reports the number of connected, non-isomorphic graphs of size up to 9 (columns). Moreover, in parenthesis, it is given for each one of such graphs the number of distinct colourings which use 2 or more colours derived by the Stirling's triangle for subsets. The last three rows report the sum of the graphs $|\widehat{\mathcal{S}}_{\mathcal{G}(n)}| = \sum_{G\in\mathcal{G}(n)} |\widehat{\mathcal{S}}_G|$, the total number of distinct colourings $\widehat{\mathcal{S}}_{\mathcal{G}(n)}$, and the size of the sample from $\widehat{\mathcal{S}}_{\mathcal{G}(n)}$ used for our experiments.

once starting from each colouring of $\widehat{\mathcal{S}}_G$.[13] In Table 4.3 we report the sum of the size of $\widehat{\mathcal{S}}_G^{\mathcal{II}}$ over all graphs of a given order $\mathcal{G}(n)$, *i.e.*, the value $|\widehat{\mathcal{S}}_{\mathcal{G}(n)}^{\mathcal{II}}| = \sum_{G\in\mathcal{G}(n)} |\widehat{\mathcal{S}}_G^{\mathcal{II}}|$. Since $\widehat{\mathcal{S}}_G^{\mathcal{II}} \subseteq \widehat{\mathcal{S}}_G$, the smaller the size of the set $\widehat{\mathcal{S}}_{\mathcal{G}(n)}^{\mathcal{II}}$ is the more feasible colourings the algorithm $\mathcal{II}$ finds, and, hence, the better it performs. Computation times for all algorithms are negligible, given the very small size of the graphs.

Observing the results of Table 4.3 the following conclusions can be drawn.

- The introduction of swaps, pair swaps, cyclic exchanges, or path exchanges ($\mathcal{N}_2$, $\mathcal{N}_3$, $\mathcal{N}_4$, or $\mathcal{N}_5$, respectively) in combination with one-exchange ($\mathcal{N}_1$) is profitable and the use of pair swaps $\mathcal{N}_3$ combined with one-exchange and swaps ($\mathcal{N}_1 \cup \mathcal{N}_2$) is preferable to the use of one-exchange combined with cyclic exchanges ($\mathcal{N}_1 \cup \mathcal{N}_4$).
- The best neighbourhood combination is $\mathcal{N}_4 \cup \mathcal{N}_5$, *i.e.*, cyclic and path exchanges.
- The use of pair swaps in combination with cyclic and path exchanges $\mathcal{N}_3 \cup \mathcal{N}_4 \cup \mathcal{N}_5$ does not bring any advantage (at least on the small graphs considered) and given that it entails an even further increase in the complexity of the algorithm it may be avoided.
- It is relevant to note that the neighbourhood $\mathcal{N}_4 \cup \mathcal{N}_5$ is able to solve only one local optimum which was not solved by the neighbourhood union of path exchanges and pair swaps ($\mathcal{N}_2 \cup \mathcal{N}_5$) while it introduces a large margin of improvement over the neighbourhood union of one-exchanges and cyclic exchanges ($\mathcal{N}_1 \cup \mathcal{N}_4$). We recall that $\mathcal{N}_4$ allows swaps and that $\mathcal{N}_5$ allows one exchanges as degenerate cases. Therefore, this result indicates that path exchanges rather than cyclic exchanges are the main reason of the large improvement due to the neighbourhood $\mathcal{N}_4 \cup \mathcal{N}_5$.

[13] Each iterative improvement algorithm $\mathcal{II}$ can be seen as a many-to-one function from the domain $\widehat{\mathcal{S}}_G$ to the set of local optima $\widehat{\mathcal{S}}_G^{\mathcal{II}}$. It takes a starting solution and returns a local optimum. In fact, given that $\mathcal{II}$ can make stochastic choices for breaking ties between solutions that are equally good, it is a multi-valued function. When we run $\mathcal{II}$ from all solutions in $\widehat{\mathcal{S}}_G$ then we are certain to obtain at least once all solutions in $\widehat{\mathcal{S}}_G^{\mathcal{II}}$. When, instead, we run it from a sample of solutions in $\widehat{\mathcal{S}}_G$, then the set of solutions found is only a subset of $\widehat{\mathcal{C}}_G^{\mathcal{II}}$.

	$\widehat{\mathcal{S}}^{\mathcal{II}_{\mathcal{N}_1}}_{\mathcal{G}(n)}$	$\widehat{\mathcal{S}}^{\mathcal{II}_{\mathcal{N}_1\cup\mathcal{N}_2}}_{\mathcal{G}(n)}$	$\widehat{\mathcal{S}}^{\mathcal{II}_{\mathcal{N}_1\cup\mathcal{N}_2\cup\mathcal{N}_3}}_{\mathcal{G}(n)}$	$\widehat{\mathcal{S}}^{\mathcal{II}_{\mathcal{N}_1\cup\mathcal{N}_4}}_{\mathcal{G}(n)}$	$\widehat{\mathcal{S}}^{\mathcal{II}_{\mathcal{N}_2\cup\mathcal{N}_5}}_{\mathcal{G}(n)}$	$\widehat{\mathcal{S}}^{\mathcal{II}_{\mathcal{N}_4\cup\mathcal{N}_5}}_{\mathcal{G}(n)}$	$\widehat{\mathcal{S}}^{\mathcal{II}_{\mathcal{N}_3\cup\mathcal{N}_4\cup\mathcal{N}_5}}_{\mathcal{G}(n)}$
3	0	0	0	0	0	0	0
4	1	1	0	1	0	0	0
5	10	10	2	9	0	0	0
6	83	76	27	58	4	4	4
7	532	484	150	312	15	15	15
8	348	315	83	175	12	11	11
9	134	117	29	52	1	1	1

Table 4.3.: For each graph order and neighbourhood structure, the table reports the size of the set $|\widehat{\mathcal{S}}^{\mathcal{II}}_{\mathcal{G}(n)}|$, that is, the number of solutions in $|\widehat{\mathcal{S}}^{\mathcal{G}(n)}|$ that are infeasible colourings and at the same time local optima. For n from 7 to 9 the value is an estimate because, given that we use a sample from $\mathcal{G}$, not all solutions in $|\widehat{\mathcal{S}}^{\mathcal{II}}_{\mathcal{G}(n)}|$ are effectively counted (this explains the reason why the reported size decreases).

4.9.3. Neighbourhood examination and heuristic rules for its speed-up

The size of the neighbourhoods $\mathcal{N}_4$ and $\mathcal{N}_5$ is exponential and therefore their use is unfeasible unless we do not find a way to examine them efficiently. In this section we describe our method. We first give account of an exact algorithm by dynamic programming proposed by Ahuja et al. (2001b) for exploring $\mathcal{N}_4$. Then we propose several ways of truncating the examination, such that the algorithm becomes an efficient heuristic. This work is a further development of the algorithm presented by Dumitrescu (2002).

The exact algorithm makes use of the idea of *subset disjoint paths*, that is, paths that visit every subset T_i of the improvement graph $G'(V', D')$ at most once (see page 107 for the definition of the improvement graph). It is clear that the path can be closed to form a subset disjoint cycle by an arc that connects the last and the first vertex of such a path. This arc always exists given the way the improvement graph is constructed.

We denote a path in the improvement graph by $p = (i_1, \ldots, i_l)$, where i_1 is the start vertex of p, denoted by $s(p)$ and i_l is the end vertex of p, denoted by $e(p)$. We associate a binary vector $w(p) \in \{0,1\}^k$ with the path p, where $w_j(p) = 1$ if and only if p visits the subset T_j. The cost of p, denoted by $c(p)$, is the total cost of the arcs in the path, *i.e.*, $c(p) = \sum_{j=1}^{l-1} c_{i_j,i_{j+1}}$.

We say that the path p_1 *dominates* the path p_2 if $s(p_1) = s(p_2)$, $e(p_1) = e(p_2)$, $c(p_1) \leq c(p_2)$, $w(p_1) \leq w(p_2)$, and the paths do not coincide. This definition will help us to discard paths that are not promising. We call a *treatment* of a path the extension of that path along all outgoing arcs. After a treatment, a path is marked as *treated*. At any time, only paths that have not previously been treated are selected for treatment. We describe the details of the algorithm for finding *subset disjoint negative cost cycles* SDNCC in Algorithm 4.4.

If the parameter `LAB_LIM` is set large enough, the algorithm SDNCC is exact. Given the correspondence between the vertices of G' and the vertices of G, the output cycle q^* contains the ordered sequence of vertices for a cyclic exchange in G and c^* is its cost.

The algorithm uses a well known lemma from Lin and Kernighan (1973) which establishes that if a sequence of edge costs has negative sum, there is a cyclic permutation of these edges such that every partial sum is negative. This allows to restrict the dynamic programming recursion as follows:

1. A label of length one is created only for negative edges.
2. A label of length larger than one is extended with a new edge only if the partial sum of the costs associated with the edges remains negative.

Function SDNCC(G'(V',D'));
Let $\mathcal{P} = \{(i,j) : (i,j) \in D', c(i,j) < 0\}$ %all negative cost paths of length 1
Mark all paths in $\mathcal{P}$ as untreated;
Initialise the best cycle $q^* = ()$ and $c^* = 0$;
for each $p \in \mathcal{P}$ **do**
  if $(e(p), s(p)) \in D'$ and $c(p) + c(e(p), s(p)) < c^*$ **then**
    q^* = the cycle obtained by closing p and $c^* = c(q^*)$;
  end
end
while $\mathcal{P} \neq \emptyset$ **do**
  Let $\widehat{\mathcal{P}} \subseteq \mathcal{P}$ be the set of the cheapest LAB_LIM untreated paths;
  $\mathcal{P} = \emptyset$;
  while $\exists\, p \in \widehat{\mathcal{P}}$ untreated **do**
    Select some untreated path $p \in \widehat{\mathcal{P}}$ and mark it as treated;
    for each $(e(p), j) \in D'$ s.t. $w_{\varphi(v_j)}(p) = 0$ and $c(p) + c(e(p), j) < 0$ **do**
      Add the extended path $(s(p), \ldots, e(p), j)$ to $\mathcal{P}$ as untreated;
      if $(j, s(p)) \in D'$ and $c(p) + c(e(p), j) + c(j, s(p)) < c^*$ **then**
        q^* = the cycle obtained by closing the path $(s(p), \ldots, e(p), j)$;
        $c^* = c(q^*)$;
      end
    end
  end
  for each $p' \in \mathcal{P}$ subject to $w(p') = w(p),\ s(p') = s(p),\ e(p') = e(p)$ **do**
    Remove from $\mathcal{P}$ the path of higher cost between p and p';
  end
end
return a negative cost cycle q^* of cost c^*

Algorithm 4.4: An algorithm that finds subset disjoint negative cost cycles.

The application of this rule does not cause the omission of any promising subset disjoint negative cost cycle.

We use Algorithm 4.4 also for searching for improving path exchanges, that is, to examine $\mathcal{N}_5$. Unfortunately, because of how we construct the improvement graph, the cost of a subset disjoint path in the improvement graph does not correspond to the cost of a path exchange in the real graph. An adjustment is required. Given the subset disjoint path $p = (i_1, \ldots, i_l)$ in G', which corresponds to $\tilde{p} = (v_{i_1}, \ldots, v_{i_l})$ in G, the evaluation function of a neighbour $\mathcal{C}'$ obtained from $\mathcal{C}$ after a path exchange given by $\tilde{p}$ is: $f(\mathcal{C}') = f(\mathcal{C}) + c(p) - |A_{C_{\varphi(v_{i_1})}}(v_{i_1})| - |A_{C'_{\varphi(v_{i_l})}}(v_{i_l})|$, where $C'_{\varphi(v_{i_l})} = C_{\varphi(v_{i_l})} \cup \{v_{l-1}\}$. In order to find the best (most improving) cyclic *or* path exchange in neighbourhood $\mathcal{N}_4 \cup \mathcal{N}_5$ we must then modify Algorithm 4.4 such that it checks also the cost of paths whenever it checks the cost of cycles. The output q^* is then the best path or cycle found, and c^* is its cost in G.

However, while the use of the lemma from Lin and Kernighan (1973) guarantees that the cyclic exchange of minimal cost will not be missed, the same is not true for path exchanges. Indeed, given that the cost of subset disjoint paths in Algorithm 4.4 does not correspond to the cost of path exchanges in G, discarding subset disjoint paths of length one that have non negative cost in the dynamic programming recursion may entail missing subset disjoint paths. In order to make the Algorithm 4.4 exhaustive also for path exchanges, we must therefore modify it to treat also subset disjoint paths of non negative

cost of any length.

The number of subset disjoint paths of various length grows exponentially and if the graph and the number of subsets of the partition of V' are large, their examination becomes quickly intractable. We decided, therefore, to study the introduction of heuristic truncation rules to limit the search.

Rule 1. Only subset disjoint negative paths are treated as from lemma of Lin and Kernighan (1973). In other terms, we trade the optimality of path exchanges in favour of the contribution of the lemma in pruning the search.

Rule 2. Only a limited number of subset disjoint paths for each length of a path is treated. This is achieved by restricting the parameter `LAB_LIM`. In this way the computational complexity of the algorithm is reduced to $\mathcal{O}(|V|^2)$. This, however, entails that the output of the algorithm will no more be necessarily an optimal solution neither for the cyclic exchanges.

Rule 3. Before running Algorithm 4.4 for finding cyclic or path exchanges, the neighbourhood $\mathcal{N}_1$ is examined, as this can be done very quickly. Then, Algorithm 4.4 is run only if the best neighbour found in $\mathcal{N}_1$ has the same quality as the current solution. The other two possible cases are treated in this way:

1. if the best neighbour in $\mathcal{N}_1$ is an improving solution, then it is accepted and the search proceeds to the next iteration;
2. if the best neighbour in $\mathcal{N}_1$ is a worsening candidate solution, then the search ends and the current solution is deemed to be a local optimum.

Empirically, we observed that situation 2 occurs much more rarely than the case in which the best solution leaves the cost unchanged.

In order to find a good setting for `LAB_LIM`, we tested the effect of different values for `LAB_LIM` on $\mathcal{II}_{\mathcal{N}_1 \cup \mathcal{N}_4}$ for some instances of the DIMACS suite. We show the behaviour obtained on two representative cases, instances `DSJC125.5` and `queen16_16`, in Figure 4.8. For each local optimum found we plot its quality, the computation time and the number of iterations needed to reach it. We observe that as `LAB_LIM` increases slightly, the number of iterations to reach local optima decreases, but the time increases strongly. The solution quality has, instead, an apparent asymptotic behaviour. The analysis suggests that a reasonable trade off between solution quality and running time is obtained for values of `LAB_LIM` between 50 and 100.

Next, we investigate the effect of the truncation rules for $\mathcal{II}_{\mathcal{N}_4 \cup \mathcal{N}_5}$ on the small size graphs solved in Section 4.9.2. We consider the following cases:

Exhaustive: No truncation rules are used and therefore the solutions returned by Algorithm 4.4 are exact.

Truncated, Variant 1: Rules 1, 2, and 3 are all active.

Truncated, Variant 2: All the three Rules are active but Rule 1 is modified such that subset disjoint paths of length larger than 1 which have cost zero are also maintained (thus reducing the omission of subset disjoint negative paths).

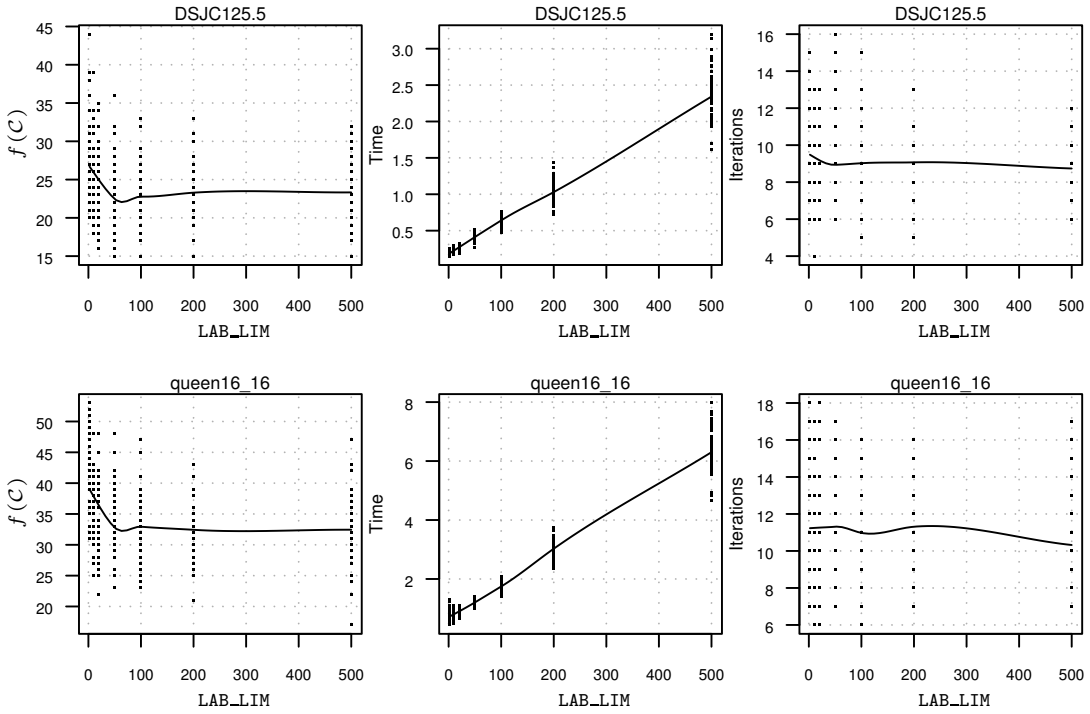


Figure 4.8.: Two representative cases to study the influence of the parameter LAB_LIM on $\mathcal{II}_{\mathcal{N}_5}$. The plots consider instances DSJC125.5 (upper now) and queen16_16 (lower now) with k fixed to 17. Reported are the median the solution quality, $f(\mathcal{C})$, the run time and the number of iterations necessary to reach a local optimum. Times are expressed in seconds. A smooth regression line is superimposed.

Truncated, Variant 3: All three Rules are active but Rule 3 is modified such that Algorithm 4.4 is run also when the best neighbour in $\mathcal{N}_1$ is a non-improving neighbour.

We report the results of our experiments in Table 4.4. In fact, we do not expect to see an impact of Rule 2 with LAB_LIM equal to 50 or 100, because the size of subset disjoint paths in Algorithm 4.4 is not very large with these graph and good paths are hardly missed. The other rules have, instead, a considerable effect.

We may draw the following conclusions. The deterioration of the results caused by the application of the heuristic truncation rules is relevant and it is mainly due to the omission of promising subset disjoint negative cost paths. The three variants considered do not introduce substantial differences in the performance. The application of the three rules in their standard version, as in Variant 1, has, therefore, to be preferred, because the contribution to the pruning of the search space is higher. Finally, complementary experiments, omitted here, confirmed the belief that the heuristic truncation rule with the strongest influence impact on a loss of solutions is Rule 1. Nevertheless, this rule has also the strongest impact on the size of the neighbourhood to explore and for graphs with large size this contribution is primarily important (as we will see in Table 4.6).

The contribution of each neighbourhood

In Table 4.5 we report the number of times moves in a particular neighbourhood are observed. For neighbourhoods obtained by the composition of other neighbourhoods we

n	Exhaustive $\lvert\widehat{\mathcal{S}}^{\mathcal{II}}_{\mathcal{G}(n)}\rvert$	Truncated, Variant 1 $\lvert\widehat{\mathcal{S}}^{\mathcal{II}}_{\mathcal{G}(n)}\rvert$	Truncated, Variant 2 $\lvert\widehat{\mathcal{S}}^{\mathcal{II}}_{\mathcal{G}(n)}\rvert$	Truncated, Variant 3 $\lvert\widehat{\mathcal{S}}^{\mathcal{II}}_{\mathcal{G}(n)}\rvert$
3	0	0	0	0
4	0	1	1	1
5	0	9	9	9
6	4	52	57	52
7	15	260	289	260
8	11	134	146	134
9	1	54	45	54

Table 4.4.: Comparison of truncation rules in the exploration of neighbourhood $\mathcal{N}_4 \cup \mathcal{N}_5$. The size of the set of distinct local optima is used to indicate the capability of solving the problem.

assume that ties between moves of equal value are broken in favour of the neighbourhood that comes first in the order. Moreover, for neighbourhoods $\mathcal{N}_4$ and $\mathcal{N}_5$ we take into account if the path exchanges and the cyclic exchanges are degenerate one-exchanges or swap exchanges, respectively. For the cyclic exchange neighbourhood $\mathcal{N}_4$ and for the path exchange neighbourhood $\mathcal{N}_5$ we also report in parenthesis the maximal length for an exchange, that is, the number of edges in the corresponding cycle or path in the improvement graph. For $\mathcal{N}_4 \cup \mathcal{N}_5$ we report the results for the exhaustive versions and for the truncated version of Variant 1.

It is evident from the table that paths are more used than cycles. In exhaustive search, the number of paths or cycles performed are less than in the truncated version, which indicates that a thorough exploration of the neighbourhood allows to reach faster (in terms of the number of moves) a feasible solution. Moreover, the comparison of the third and fourth column (that is, if path exchanges are examined before cyclic exchanges) indicates that many situations solved by cyclic exchanges can be solved also by path exchanges (see results relative to the row for $\mathcal{N}_4$). We add that introducing $\mathcal{N}_3$ in $\mathcal{N}_4 \cup \mathcal{N}_5$ was useless because no move from $\mathcal{N}_3$ was then taken (this confirms the result discussed in Section 4.9.2).

A final note is on the length of cyclic and paths exchanges accepted during the iterative improvement process. Even with the small graphs presented here, with exhaustive search cyclic exchanges change colours to even 4 vertices at a time and path exchanges to 3. This result is relevant because it gives further evidence that searching well beyond the first neighbourhood is profitable, as an improving exchange can be found only moving many vertices. If truncation rules are adopted, the length of the chosen cycles and paths increases because easier (*i.e.*, shorter) exchanges may be missed.

Larger graphs

We already mentioned that according to an important result of Erdős (1961) the chromaticity of a graph appears as a global phenomenon rather than being only locally characterised. What exactly causes high chromaticity still remains a mystery (Diestel, 2000). This suggests that results on small graphs are not enough to predict a good behaviour of an algorithm. We therefore applied the iterative improvement algorithms also to larger random graphs, in particular the instances `DSJC125.`$*$, from the DIMACS benchmark suite.

The size of these instances still makes feasible an exhaustive examination of the neighbourhoods. The chromatic number of these graphs is still unknown; then we vary k

	n	$\mathcal{II}_{\mathcal{N}_1\cup\mathcal{N}_4}$ exhaustive	$\mathcal{II}_{\mathcal{N}_2\cup\mathcal{N}_5}$ exhaustive	$\mathcal{II}_{\mathcal{N}_4\cup\mathcal{N}_5}$ exhaustive	$\mathcal{II}_{\mathcal{N}_5\cup\mathcal{N}_4}$ exhaustive	$\mathcal{II}_{\mathcal{N}_4\cup\mathcal{N}_5}$ truncated
$\mathcal{N}_1$	3	5	5	5	5	5
	4	55	55	55	55	49
	5	462	462	462	462	408
	6	5094	5094	5094	5094	4058
	7	74119	74122	74122	74122	54859
	8	85023	85024	85024	85024	60255
	9	95641	95641	95641	95641	66301
$\mathcal{N}_2$	3	–	–	–	–	–
	4	–	–	–	–	3
	5	–	–	–	–	20
	6	7	7	7	7	241
	7	55	55	55	55	2843
	8	33	33	33	33	2566
	9	17	17	17	17	2266
$\mathcal{N}_4$	3	–	–	–	–	–
	4	–	–	–	–	–
	5	1(3)	–	1(3)	–	3(3)
	6	19(3)	–	19(3)	–	138(4)
	7	197(3)	–	197(3)	–	3036(4)
	8	141(4)	–	142(4)	1(3)	4042(5)
	9	65(4)	–	65(4)	–	4541(6)
$\mathcal{N}_5$	3	–	–	–	–	–
	4	–	1(2)	1(2)	1(2)	–
	5	–	10(2)	9(2)	10(2)	5(2)
	6	–	81(2)	62(2)	81(2)	165(3)
	7	–	562(3)	365(3)	562(3)	3916(4)
	8	–	306(2)	165(2)	306(2)	5690(4)
	9	–	116(3)	51(3)	116(3)	7536(5)

Table 4.5.: Number of moves chosen from each neighbourhood. In parenthesis the maximal length registered for cyclic and path exchanges (note that a cyclic exchange of length 3 corresponds to $\{v_1, v_2, v_3, v_1\}$ and entails that 3 vertices change colours, while a path exchange of length 3 corresponds to $\{v_1, v_2, v_3, v_4\}$ and also entails that 3 vertices change colours, as v_4 serves only to determine the arrival colour for v_3.

starting from a reasonable high value down to the best known value in the literature. At each k we run an Iterative Improvement starting from 1000 distinct initial colourings which are maintained the same over all algorithms.

The results are given in Table 4.6 for different neighbourhoods. The performance is identified by the rate of success in finding feasible colourings starting from the 1000 initial colourings (%), by the median number of conflicting edges in the final infeasible colourings ($\widetilde{f}$), and by the median CPU time for each single run of Iterative Improvement in seconds. Better performance in the quality of the final solution is indicated in the first place by a higher success rate and in the second place by a low $\widetilde{f}$.

These results confirm those for the small graphs. In particular, the Iterative Improvement with the exhaustive examination of $\mathcal{N}_4 \cup \mathcal{N}_5$ clearly outperforms all other alternatives; however the CPU times are the largest and limit the field of applicability of this choice to graphs of limited size. The results of the truncated versions are, however, interesting since they remain competitive with the exhaustive examination, above all on graphs of small density and on high density graphs when the value of k is low. Moreover their running time is only about 10% of the time of the exhaustive methods. Finally, among the three variants of the truncation rules, we have confirmation that Variant 1 is the best choice as the other two have the only effect of worsening time performance.

In conclusion, we gave enough evidence to justify the focus in the remainder of the chapter to only one version of very large scale neighbourhood: the structure $\mathcal{N}_4 \cup \mathcal{N}_5$ with examination pruned by heuristic rules as indicated by Variant 1. The study so far

ρ	k	$II_{\mathcal{N}_1}$			$II_{\mathcal{N}_1 \cup \mathcal{N}_2}$			$II_{\mathcal{N}_4 \cup \mathcal{N}_5}$ Exhaustive			$II_{\mathcal{N}_4 \cup \mathcal{N}_5}$ Variant 1			$II_{\mathcal{N}_4 \cup \mathcal{N}_5}$ Variant 2			$II_{\mathcal{N}_4 \cup \mathcal{N}_5}$ Variant 3		
		%	$\widetilde{f}$	sec.	%	$\widetilde{f}$	sec.	%	$\widetilde{f}$	sec.	%	$\widetilde{f}$	sec.	%	$\widetilde{f}$	sec.	%	$\widetilde{f}$	sec.
0.1	5	0	23	0	0	18	0	0	13	165	0	13	15	0	13	61	0	12	63
	6	0	10	0	0	6	0	12	2	152	12	2	12	14	2	48	13	2	49
	7	2	4	0	12	2	0	88	1	134	87	1	5	91	1	37	90	1	37
	8	29	1	0	64	1	0	100	1	119	100	1	3	100	1	29	100	–	30
	9	74	1	0	93	1	0	100	–	107	100	–	1	100	–	26	100	–	26
	10	93	1	0	99	1	0	100	–	99	100	–	0	100	–	23	100	–	23
0.5	17	0	36	1	0	31	1	0	17	232	0	17	19	0	17	61	0	16	66
	18	0	29	1	0	25	1	0	11	225	0	11	19	0	11	61	0	11	65
	19	0	23	1	0	19	1	0	6	220	0	6	17	0	6	58	0	6	64
	20	0	18	1	0	14	1	12	3	213	6	3	15	3	3	55	3	3	60
	21	0	14	1	0	11	1	56	1	204	33	1	12	16	2	53	16	2	56
	22	0	11	1	0	8	1	87	1	193	61	1	10	33	1	50	33	1	55
	23	0	8	1	0	6	1	96	1	180	81	1	8	53	1	49	53	1	54
	24	0	6	1	2	4	1	99	1	174	90	1	6	68	1	47	69	1	50
	25	1	4	1	7	3	1	100	1	166	96	1	4	82	1	46	81	1	49
0.9	30	0	77	1	0	70	1	0	55	242	0	55	28	0	55	8	0	55	95
	31	0	72	1	0	64	1	0	50	238	0	50	26	0	50	8	0	50	96
	32	0	66	1	0	59	1	0	45	238	0	45	26	0	45	81	0	45	96
	33	0	62	1	0	54	1	0	40	238	0	40	27	0	40	86	0	40	104
	34	0	57	1	0	49	1	0	36	237	0	36	26	0	36	89	0	36	108
	35	0	53	1	0	45	1	0	32	237	0	32	27	0	32	91	0	32	111
	36	0	49	1	0	41	1	0	28	233	0	28	27	0	28	93	0	28	114
	37	0	44	1	0	37	1	0	25	231	0	25	26	0	25	94	0	24	117
	38	0	41	1	0	34	1	0	21	229	0	21	26	0	22	94	0	22	118
	39	0	37	1	0	31	1	0	18	225	0	18	26	0	19	96	0	19	119
	40	0	34	1	0	29	1	0	16	225	0	16	24	0	17	94	0	17	120
	41	0	31	1	0	26	1	0	13	219	0	13	23	0	15	95	0	15	122
	42	0	29	1	0	24	1	0	11	218	0	11	23	0	12	95	0	12	124
	43	0	26	1	0	22	1	0	9	242	0	9	23	0	10	99	0	10	130
	44	0	25	1	0	20	1	0	7	247	0	7	24	0	9	97	0	9	131
	45	0	23	1	0	18	1	0	5	249	0	5	24	0	8	100	0	8	133
	46	0	21	1	0	16	1	6	3	195	1	3	24	0	6	100	0	6	135
	47	0	20	1	0	14	1	18	2	185	4	3	23	0	5	100	0	5	137
	48	0	19	1	0	12	1	35	2	184	10	2	21	1	5	101	1	5	140
	49	0	17	1	0	11	1	50	2	176	19	2	20	2	4	100	2	4	141
	50	0	15	1	0	9	1	63	1	172	22	2	19	3	3	102	4	3	142

Table 4.6.: Results on the instances `DSJC125` with edge density: 0.1, 0.5, 0.9. The table reports for each of the three graphs, at different k values, the performance for solving the relative k-colouring problem. The indicators for the performance are: the % rate of success in finding a feasible colouring; the median number of conflicting edges $\widetilde{f}$; and the median CPU time expressed in hundredth of seconds to complete a single run.

was limited to iterative improvement with best improvement strategy. As the next step we will test the use of the cyclic and path exchange neighbourhood in more complex SLS algorithms and give an in-depth experimental comparison to state-of-the-art algorithms.

4.10. Stochastic Local Search algorithms

Simple Iterative Improvement algorithms typically show poor performance for the GCP when compared with more complex Stochastic Local Search algorithms. Our analysis focuses ultimately on the latter methods and in what follows we describe known and new SLS algorithms that we studied. The methods are divided into the same three different solution approaches that we discussed for Iterative Improvement: solving a sequence of k colouring problems, varying the number of colours used, and extending partial colourings. In this latter approach we consider only one algorithm that tries to exploit the potential of exhaustive techniques.

In the presentation, we dedicate a subsection for each algorithm to explain the main rules of the algorithm and the parameter settings. The general policy adopted is to give preference to algorithms with fewer parameters and, where possible, to link parameters to instance features or adjust them dynamically. Besides this, we also gave preference to the robustness of the algorithm thus allowing them to use only one parameter setting

over all instances. For known algorithms, preference is given to the parameters suggested by the respective authors. For new algorithms, instead, preliminary experiments are performed for tuning the corresponding parameters. We do not go into the details of these preliminary experiments. The methodology described in Chapter 6 could be used for such a task. Nevertheless, we point out the cases where the performance is highly variable with respect to algorithm's parameters, trying to shed light on their relationship with the features of the graphs.

4.10.1. Solving a sequence of k colouring problems

SLS algorithms in this class solve the chromatic number problem by maintaining k fixed and decreasing it each time a feasible colouring for that k is reached. We start all algorithms in this class from the number of colours k^{RLF} returned by the RLF heuristic. When a feasible k-colouring is found, then a new colouring with $k-1$ colours is searched in the next stage. We distinguish two main options of how to produce the new, possibly infeasible, $(k-1)$-colouring:

1. construct a colouring from scratch, or
2. uncolour the vertices assigned to one selected colour and re-colour them by using any of the remaining colours.

The first option is practicable with any of the construction heuristics described in Section 4.7 by setting a bound to the number of usable colours and removing the condition that the final colouring must be feasible. The construction heuristics can also be used for the second option by using the rules inherent to the heuristic for re-assigning uncoloured vertices.

Tuning. We experimentally compared these two options. For both we considered the three construction heuristics and, in addition, restricted to the second option, a random re-distribution of vertices. The comparison was carried out embedding the different alternatives in the Tabu Search algorithm on the $\mathcal{N}_1$ neighbourhood that we will describe in the next paragraph. We observed that the second option entails a faster search, most probably because it provides an initial solution of better quality for the next stage. Among the four different ways of reconstructing a partition, no significant difference could be found between a random re-distribution of vertices and a reassignment by means of the RLF heuristic, while these two variants perform significantly better than a re-colouring using the ROS or the DSATUR heuristic. Hence, for the algorithms we present next, we selected the simplest method, which is the random reassignment of colours to vertices.

Tabu Search with one-exchange neighbourhood

Tabu Search with one-exchange neighbourhood ($\mathsf{TS}_{\mathcal{N}_1}$) has been widely studied for graph colouring (Hertz and de Werra, 1987; Fleurent and Ferland, 1996; Dorne and Hao, 1998a; Galinier and Hao, 1999). In our re-implementation of this method we comply with the last reference. At each local search iteration, $\mathsf{TS}_{\mathcal{N}_1}$ chooses a best non-tabu neighbouring candidate solution from the $\mathcal{N}_1$ neighbourhood. If the colour class of vertex v changes

from C_i to C_j, it is forbidden to assign colour i to vertex v in the next *tt* steps. This prohibition is broken only when such a move would lead to an improvement over the best candidate solution encountered so far (aspiration criterion). The tabu tenure, *tt*, for a specific vertex–colour pair (v, i) is set proportional to the size of the neighbourhood as $tt = random(10) + \delta \cdot 2 \cdot |E_V^c|$ where $random(10)$ is an integer uniformly chosen from $\{0, \ldots, 10\}$. Hence, the tabu tenure varies dynamically at run-time in dependence of the evaluation function value (similar dynamic tabu tenures are used by Dorne and Hao, 1998a while they are not present in the original implementation of Hertz and de Werra, 1987). To avoid forbidding too many moves, *tt* should be smaller than the neighbourhood size. This can be achieved by setting δ to a value that is much smaller than k. We code the tabu list as a matrix $k \times |V|$ and in each cell we record the iteration number until which the move stays tabu. Finally, if more than one move produces the same effect on the evaluation function, one of these is selected randomly.

Tuning. Based on some preliminary experimental investigation, described in Chiarandini and Stützle (2002), we fixed δ to 0.5; the particular value of δ may affect performance but 0.5 seems to be sufficiently robust.

Tabu Search with very large scale neighbourhood

We saw in Section 4.9 that a best improvement local search with cyclic and path exchange neighbourhood outperforms the one-exchange neighbourhood from the solution quality perspective. However, as also observed in Chiarandini et al. (2003), the use of SLS methods enhances considerably the performance attainable with the use of a one-exchange neighbourhood, such that an algorithm like $\mathsf{TS}_{\mathcal{N}_1}$ becomes highly competitive.

To examine fairly the effectiveness of the cyclic and path exchange neighbourhood we embed it into an analogous Tabu Search algorithm as the one presented above. We call this algorithms $\mathsf{TS}_{\mathsf{VLSN}}$. At each step, the examination of the neighbourhood $\mathcal{N}_4 \cup \mathcal{N}_5$ follows the scheme of Variant 1 in Section 4.9.3 but embedded in a Tabu Search mechanism. More specifically, first the best non-tabu move in $\mathcal{N}_1$ is selected in the very same way as $\mathsf{TS}_{\mathcal{N}_1}$. If the move improves $f(\mathcal{C})$, where $\mathcal{C}$ is the current colouring, then the move is applied and the tabu list is updated as in $\mathsf{TS}_{\mathcal{N}_1}$. If the move is a non-worsening move, then the neighbourhood $\mathcal{N}_4 \cup \mathcal{N}_5$ is searched using Algorithm 4.4. The algorithm is slightly modified in order to consider a candidate list, that is, only paths and cycles which include at least one vertex involved in a conflict. This is achieved by including in the initial $\mathcal{P}$ only paths of length one where at least one of the two vertices is in V_c. The Tabu Search mechanism is an extension of $\mathsf{TS}_{\mathcal{N}_1}$: a path or cyclic exchange is discarded if it entails the reassignment of a recently used colour to at least one vertex. The tabu list is updated by considering the path or the cyclic exchange as a composition of one-exchanges and the tabu tenure is chosen dynamically using the same formula as for $\mathsf{TS}_{\mathcal{N}_1}$. Yet, contrarily to $\mathsf{TS}_{\mathcal{N}_1}$, $\mathsf{TS}_{\mathsf{VLSN}}$ does not use an aspiration criterion. This allows to discard vertices that are tabu very soon in the search for subset disjoint paths in Algorithm 4.4.

Tuning. We use `LAB_LIM` fixed to 50 for the reasons explained in Section 4.9.3. Moreover, the choice not to use an aspiration criterion was suggested by experiments that showed that its inclusion would only increase the computation time without improving significantly the final solution quality. The δ parameter in the definition of the tabu tenure is maintained at 0.5 as for $\mathsf{TS}_{\mathcal{N}_1}$.

Min-Conflicts heuristics

A variant of Tabu Search on the $\mathcal{N}_1$ neighbourhood is suggested by the Min-Conflicts heuristic which has successfully been applied to the CSP (Minton et al., 1992). We adapted this method to graph colouring as follows. The selection of a move is done in a two-stage selection scheme. In the first step, a vertex $v \in V^c$ is chosen randomly according to a uniform distribution. In the second step, a colour is assigned to v that minimises the number of conflicts; ties are broken randomly. This selection method is then integrated into a Tabu Search mechanism by allowing in the second stage only colours that are not tabu according to the same criterion as in $\mathsf{TS}_{\mathcal{N}_1}$. This strategy of neighbourhood examination reduces the complexity of examining the neighbourhood from $\mathcal{O}(k|V|)$ to $\mathcal{O}(k)$, when using appropriate data structures. It also reduces the chances of cycling due to the random choices involved, and therefore the tabu list can be shorter. We call this Tabu Search extension of the Min-Conflicts heuristic $\mathsf{MC\text{-}TS}_{\mathcal{N}_1}$. It was previously studied on few instances of graph colouring by Stützle (1998). Differently from that implementation in our $\mathsf{MC\text{-}TS}_{\mathcal{N}_1}$ a colour change is always forced: in case no possible colour is available one is chosen randomly. Experimental analysis suggested that this is a profitable rule.

Tuning. A static tabu tenure of 2 was selected after some preliminary experimental analysis that confirmed the tuning proposed by Stützle (1998).

Novelty$^+$

The Novelty$^+$ algorithm is a variant of the Novelty algorithm introduced by McAllester et al. (1997) for solving the satisfiability problem and it constitutes a high-performing algorithm for this problem (Hoos and Stützle, 2000). In our adaptation of Novelty$^+$ to graph colouring, which we call Nov^+, the examination of the neighbourhood $\mathcal{N}_1$ is carried out as follows. With a probability wp, where wp is a parameter, a neighbouring solution is chosen completely at random (*i.e.*, not limiting changes to involve vertices in V^c); with a probability $1 - wp$ a neighbourhood examination scheme, similar to the one in $\mathsf{MC\text{-}TS}_{\mathcal{N}_1}$, is adopted. Differently from $\mathsf{MC\text{-}TS}_{\mathcal{N}_1}$, the colour to be assigned to the vertex v in the neighbourhood examination is chosen as follows. If there is only one colour providing the best improvement in the evaluation function and this colour is not the most recent colour assigned to the vertex among all the possible colours, then it is always selected. Otherwise, it is only selected with a probability of $1 - p$, where p is a parameter called "noise setting". In the remaining cases, the colour providing the second best improvement is selected. If there are several colours which provide the best improvement, one, which is not the most recent for the vertex, is randomly selected. This selection process is also represented by a probabilistic decision tree in Figure 4.9.

Note that in the Novelty$^+$ concept a probability $p > 0$ yields the same or a similar effect as the tabu list in Tabu Search, avoiding to cycle on the same solutions. The introduction of wp, instead, prevents the algorithm to reach stagnation behaviour, as empirically shown by Hoos (1999a).

Tuning. In setting the parameters, we followed partially the indications reported in the literature and fixed wp to 0.01 (Hoos and Stützle, 2000). We tuned, instead, the parameter p setting it equal to 0.3.

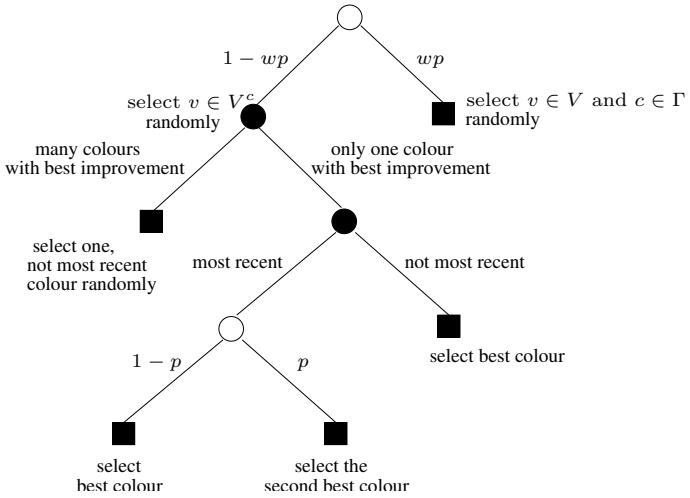


Figure 4.9.: Decision tree representation of the Novelty$^+$ mechanism for selecting a neighbour $\mathcal{C}'$ in $\mathcal{N}_1(\mathcal{C})$. Deterministic and probabilistic choices are represented by black and white circles, respectively; edges are labelled with the respective conditions and probabilities. Black boxes indicate decision actions.

Guided Local Search

Guided local search is an SLS method that tries to escape from local optima through modifications of the objective function (see Section 2.4.3 page 27). In its application to the GCP, which we denote GLS, we define an augmented evaluation function $f'(\mathcal{C}) : \mathcal{C} \to \mathbb{R}$, which consists in the original objective function $f(\mathcal{C})$ plus weights w_i associated to the edges that cause a conflict. Formally,

$$f'(\mathcal{C}) = f(\mathcal{C}) + \lambda \cdot \sum_{i=1}^{|E|} w_i \cdot I_i(\mathcal{C})$$

where $|E|$ is the number of edges, w_i is the penalty cost associated to edge i, and $I_i(\mathcal{C})$ is an indicator function that takes the value 1 if edge i causes a colour conflict in $\mathcal{C}$ and 0 otherwise. The parameter λ determines the influence of the penalties on the augmented cost function.

The penalty weights are initialised to 0 and are updated each time Iterative Improvement reaches a local optimum in f'. The modification of the penalty weights is done by first computing for each violated edge a utility

$$u_i = I_i(s) \cdot \frac{1}{1 + w_i}.$$

and then incrementing the penalties of all edges with maximal utility. We implement GLS with an iterative improvement in $\mathcal{N}_1$. At each iteration of the local search, the best neighbour according to f' is selected. Once a local optimum is reached, the search continues for sw non-worsening exchanges (side walk moves) before the evaluation function f' is updated. The update of w_i and f' is done in the worst case in $O(k|V|^2)$. The parameters to tune are λ and sw.

Tuning. The experimental analysis indicated in $\lambda = 1$ and $sw = 20$ a good setting for these two parameters. We mention also that we tested in detail two other variants of

Dynamic Local Search: (i) a dynamic local search algorithm inspired by Hutter et al. (2002), which implements the same idea of modifying the evaluation function based on probabilistic decisions rather than on indications coming from an utility function, and (ii) a further feature for Guided Local Search suggested by Mills and Tsang (2000), in which weights are scaled by $4/5$ every 200 runs of the inner Iterative Improvement. However, these two variants did not perform significantly better than the described GLS, and we decided to omit them because of their higher number of parameters. The application of GLS to the GCP is an original contribution of this thesis.

Iterated Local Search

In an attempt to enhance further the performance of $\mathsf{TS}_{\mathcal{N}_1}$, we used it as a sub-procedure for an Iterated Local Search method. A first application of ILS to the GCP is due to Paquete and Stützle (2002b). Here, we consider an improved version, introduced in Chiarandini and Stützle (2002). For the description of the algorithm, we still have to define the perturbation and the acceptance criterion.

On the basis of preliminary studies (Chiarandini and Stützle, 2002), we decided for the following configuration of ILS.

Perturbation: a number k_r, $k_r < k$, of colour classes is randomly chosen and all the vertices in the k_r colour classes are re-distributed into other colour classes by means of the ROS heuristic bounded by k, avoiding to re-insert a vertex into the same colour class in which it was previously.

Tabu Search: After each perturbation, $\mathsf{TS}_{\mathcal{N}_1}$ is run until the best solution found does not change for l_{LS} iterations. The tabu list is emptied before applying the perturbation and exchanges caused by the perturbation are inserted in the tabu list.

Acceptance Criterion: The solution to which the perturbation is applied is always the best (infeasible) colouring found so far.

Tuning. According to preliminary experiments, we set $k_r = \lfloor 0.35 \cdot k \rfloor$ and $l_{LS} = 5 \cdot |V^c| \cdot k$.

Hybrid Evolutionary Algorithm

The hybrid Evolutionary Algorithm (HEA) of Galinier and Hao (1999) has shown remarkable results for large GCP benchmark instances. HEA starts with an initial population P of solutions and cycles through stages of recombination and local search (no mutation is applied). In each iteration, two partitions are randomly chosen from P and recombined by means of the greedy partition crossover (GPX). GPX generates from two candidate partitions $\mathcal{C}_1 = \{C_1^1, \ldots, C_k^1\}$ and $\mathcal{C}_2 = \{C_1^2, \ldots, C_k^2\}$ a new partition by alternatingly selecting colour (sub)classes of each parent. At step i of the crossover operator, $i = 1, \ldots, k$, GPX chooses in parent $\mathcal{C}_1$ (if i is odd) or in parent $\mathcal{C}_2$ (if i is even) a colour class with the maximal number of vertices to become colour class C_i of the offspring; after each step, the vertices in the set C_i are removed from both parents. The vertices that remain unassigned are then added to a randomly chosen partition. The new partition returned by GPX is then improved by $\mathsf{TS}_{\mathcal{N}_1}$ and is inserted in the population P replacing the worse of the two parents.

Tuning. A problem with the computational results of HEA presented in Galinier and Hao (1999) is that the number of iterations for $\mathsf{TS}_{\mathcal{N}_1}$ is fine-tuned not only on a specific instance but even to the specific value k. In our re-implementation of HEA we avoid this. We tested two alternatives: either fixing the number of iterations of $\mathsf{TS}_{\mathcal{N}_1}$ to 1000 or to 10000. Experiments clearly indicate that using 10000 iterations for $\mathsf{TS}_{\mathcal{N}_1}$ is the best choice. The other relevant differences are the following (note that they are all tested improvements over the original version in the context of the chromatic number problem). RLF is used to determine the initial number k_{RLF} of colours. Hence, HEA starts from the same number of colours as the other algorithms. The initial population is generated using the DSATUR heuristic bounded to the current k and each time a feasible colouring is reached, the population is re-initialised from scratch for $k-1$. The population is re-initialised if the average distance between colourings in the population falls below a threshold of 20. Note that the exact distance in the $\mathcal{N}_1$ neighbourhood between pairs of colourings is the *partition distance* and can be computed in polynomial time by solving a weighted bipartite matching problem, also known as linear sum assignment problem (Gusfield, 2002; Glass and Prügel-Bennett, 2005). For this task we used the publically available shortest augmenting path algorithm (Jonker and Volgenant, 1987) that works fast for the size of the assignment instances we face (the size is determined by k).

Recently, Galinier et al. (2002) presented an adaptive memory approach based on the same ideas of HEA but which uses only a single solution instead of a population of solutions. This algorithm, however, does not improve on the performance of HEA and it remains unclear, whether it is more robust with respect to parameter settings; therefore, we do not consider it here.

4.10.2. Varying the number of used colours

Simulated Annealing with Kempe chain neighbourhood

Johnson et al. (1991) studied different variants of Simulated Annealing algorithms. We re-implement the most promising, which is based on the Kempe chain neighbourhood ($\mathsf{SA}_{\mathcal{N}_6}$). $\mathsf{SA}_{\mathcal{N}_6}$ uses the evaluation function $g(s) = -\sum_{i=1}^{k}(|C_i|)^2$. At each step of $\mathsf{SA}_{\mathcal{N}_6}$, a neighbouring solution is generated in three steps:

Step 1: select a non-empty colour class C_i, a vertex $v \in C_i$, and a non-empty colour class C_j uniformly at random, such that C_i and C_j do not form a full Kempe chain;

Step 2: determine the Kempe chain K_{ij} of colour classes C_i and C_j that contains vertex v.

Step 3: swap colours i and j for all vertices in K_{ij}.

The neighbour is accepted if it improves over the current solution; otherwise it is accepted with a probability that depends on the deterioration in the evaluation function and a control parameter T called temperature. From Section 2.4.3 this probability is

$$p_{accept}(T, \mathcal{C}, \mathcal{C}') = \begin{cases} 1 & \text{if} \quad f(\mathcal{C}') \leq f(\mathcal{C}) \\ e^{-\frac{f(\mathcal{C}')-f(\mathcal{C})}{T}} & \text{otherwise} \end{cases}$$

While in the original version the initial solution of $\mathsf{SA}_{\mathcal{N}_6}$ is generated by the ROS heuristic, here we use the initial colouring of RLF. $\mathsf{SA}_{\mathcal{N}_6}$ uses a geometric cooling schedule,

which is defined by (i) the initial temperature T_i; (ii) the number of iterations to be spent at a certain temperature determined by either a *cutoff* parameter, Tc, that imposes a limit on the number of accepted neighbours, or by a *temperature length*, Tl, that imposes a limit on the maximal number of iterations at that temperature; (iii) the decrease of the temperature by a standard geometric cooling: $T_{i+1} = \rho \cdot T_i$; and (iv) a termination condition, which in our case is the computation time (in the original $\mathsf{SA}_{\mathcal{N}_6}$ a different termination condition is used). The values of Tl and Tc are related to the size of the instance. In particular, $Tl = \epsilon \cdot |V| \cdot k$ and $Tc = \kappa \cdot |V| \cdot k$, where ϵ and κ are two parameters.

Tuning. The tuning of $\mathsf{SA}_{\mathcal{N}_6}$ exhibits a similar problem as HEA, namely, in Johnson et al. (1991) different settings were used for different computation times and different final solutions. In order to find the best compromise with our time limits, we run preliminary experiments. Based on these experiments we could verify that the parameters used in most of the cases by the original version to obtain peek performance do not yield runtimes that exceed our time limit. We set, therefore, $Ti = 5$, $\kappa = 0.1$, $\rho = 0.983$ and $\epsilon = 16$.

We also developed a Tabu Search approach similar to $\mathsf{TS}_{\mathcal{N}_1}$ that works with k variable. It performed, however, very similarly to $\mathsf{SA}_{\mathcal{N}_6}$, therefore we remove it from this final analysis.

4.10.3. Extending partial colourings: a semi-exhaustive approach

Next, we introduce a parametrised version of the RLF heuristic that exploits, where possible, the potential of exhaustive search.

XRLF

The XRLF algorithm was also introduced by Johnson et al. (1991) and appeared to give competitive results when relative long time was allowed. It exploits the possibilities offered by exhaustive search. We saw in Section 4.6 that BB-GCP can solve in short time graphs with up to 100 vertices. The concept of XRLF is to use this algorithm after having reduced the graph to a reasonable size. Reducing the graph is done by removing independent subsets of V, trying at the same time to minimise the number of edges in the residual graph. This procedure is a parametrised generalisation of RLF, which uses ideas suggested by Johri and Matula (1982). In particular, RLF is modified in order to find independent sets C contained in the current set V' of uncoloured vertices such that the size of C is maximised and the number of edges in the residual graph, $V' \setminus C$ is minimised. The construction proceeds by selecting, from a random sample from V' of `CANDNUM` candidates, a vertex adjacent to the maximal number of uncoloured non-candidates. If the number of remaining candidates for C is less than `SETLIM`, the construction is done by exhaustive search with the procedure given in Algorithm 4.5. This construction process is then repeated for `TRIALNUM` number of times, taking the best result. It stops when the number of uncoloured vertices becomes smaller than a parameter `EXACTLIM`. At this point, the remaining vertices are coloured with BB-GCP. Although we saw in Section 4.6 that Ex-DSATUR is more robust than BB-GCP on graphs with small density, for conformity with the original version, we continue to use BB-GCP. Further details on XRLF can be found in the original paper.

```
Function Independent_Set(G(V,E));
return Find_Set(G,V,∅,∅,0,0);

Function Find_Set(G(V,E),U,C,C*,c,best);
while |U| > 0 do
    select some v ∈ U and let U = U \ {v};
    T = {u | u ∈ U and {u,v} ∉ E};
    c_v = |{{u,v} ∈ E : v ∈ V \ C}|;
    if (c + c_v > best) or (c + c_v = best and |C*| < |C ∪ {v}|) then
        C* = C ∪ {v} and best = c + c_v;
    end
    C* = Find_Set(G,T,C ∪ {v},C*,c + c_v,best);
end
return C*
```

Algorithm 4.5: An algorithm for finding an independent set such that the number of edges in the remaining graph is minimised. C is a current independent set, C^* is the best independent set found, and U is the set of vertices that can still become member of C.

The tuning of `SETLIM`. When the current set V' of uncoloured vertices is smaller than `SETLIM`, the exact procedure Independent_Set of Algorithm 4.5 is used to find an independent set $C^* \subseteq V'$ such that $|\{\{u,v\} \in E : u \in C^* \text{ and } v \in V' \setminus C^*\}|$ is maximum. We report here the procedure because it was not given in the original paper. The efficiency of this algorithm is important because it is run `TRIALNUM` times for almost all colour classes in the graph and the performance of XRLF improves `TRIALNUM` is kept high.

The characteristics of the subgraph V' have a strong influence on the computational cost of Algorithm 4.5. In Figure 4.10, left, we show the growth in time of Algorithm 4.5 for random graphs of different size and density. The search in graphs of low density is very slow. This is reasonable considering that in the extreme case of a graph consisting of n isolated nodes without any edge, the number of possible independent sets grows at the order of 2^n. In contrast, graphs with high density are searched almost instantly, since for increasing density the number of possible independent sets tends to zero. If we fix the computational cost for solving V' to 5 seconds on our usual machine we obtain by linear regression the empirical formula $|V'| = (15.5\rho_{V'} + 4.7)^2$. Accordingly, for sizes of the graphs V' below the indicated threshold, the computation time of Algorithm 4.5 will be below 5 seconds. The value of `SETLIM` can therefore be linked to the density of the graph V' and tuned automatically. Experimental observations revealed that this is not enough to predict precisely the computational cost of Algorithm 4.5. Graphs in which the vertex degree is highly variable also create problems. Unfortunately, running the function Independent_Set on the random graphs of Figure 4.1, varying the size of the graph among 30, 50 and 100 vertices was not helpful for deriving a better predictive model.

In order to understand the reason for the uncommonly high computation time of Independent_Set we plotted some of the graphs that originate this behaviour and looked at the distributions of the vertex degree. In Figure 4.11, we show two graphs of size 40 with almost the same density and range characteristics. The graph on the left is a Geometric graph with edge density 0.5 and it is solved very quickly by the algorithm. The graph on the right is, instead, one of the graphs that requires long computation time. The histograms relative to the two graphs indicate that hard graphs are those with degree distributions that exhibit two peaks (modes) in correspondence to the two extremes. In

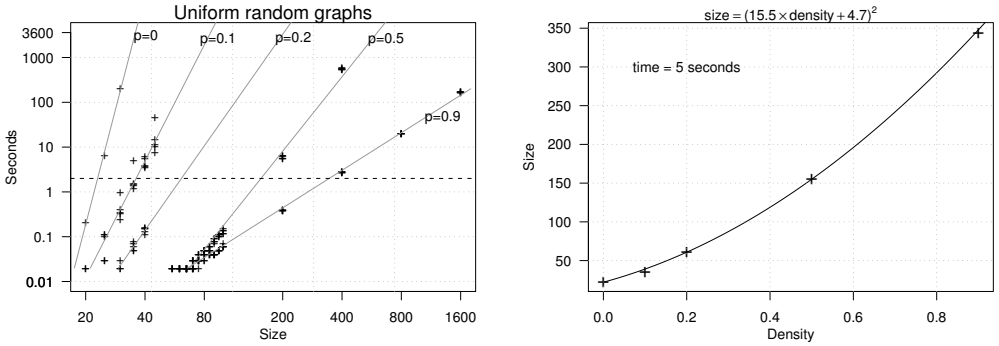


Figure 4.10.: On the left, the variation of computation time for Algorithm 4.5 on Uniform random graphs of diverse size (x-axis) and edge probability p. On the right, the curve of graphs that yield a computation time of 5 seconds.

these cases, the number of possible independent subsets is high. Unfortunately, we were unable to find a measure with low computational cost to detect these "pathological" distributions. The empirical solution that we adopted is to limit the use of Independent_Set to graphs of normalised vertex degree range below 0.2 (besides the size restriction of SETLIM in $(0, (15.5\rho_{V'} + 4.7)^2]$). This upper bound on the range eliminates the "pathological" graphs but also some other graphs which would be easily solved. Finally, for densities lower than 0.1 we run Algorithm 4.5 only if the size of V' is smaller than 25.

Eliminating the important parameter SETLIM allows a better control on the computation time of XRLF. Nevertheless, for the instances of the benchmark suite we still had to adjust CANDNUM and TRIALNUM on each individual instance. Finally, also EXACTLIM needs to be adjusted. The set of parameters adopted for the instances of the DIMACS benchmark suite as well as with the randomly generated graphs that we introduce later, are reported in Appendix C.1.2 of the thesis. Unfortunately trying to prevent XRLF to use more than the time allowed we ended up having that, on many instances, XRLF, actually, uses much less than the time allowed. Therefore, for the graphs on which results were published, we comply with the parameters indicated by the original paper. We point out, however, that the highly unpredictable behaviour of XRLF is a strong argument against its use.

XRLF is based on the idea of first removing independent sets and then colouring a reduced graph. An analogous idea has been used by other researchers substituting the two exact algorithms that compose XRLF with approximate SLS algorithms. This concept was already used by Hertz and de Werra (1987) whose paper is actually antecedent to the introduction of XRLF. Similar examples are also provided by Fleurent and Ferland (1996) and Dorne and Hao (1998a) who apply their newly introduced algorithms, mainly based on hybridisations of Tabu Search and Genetic Algorithm, after removing independent sets from the graph. This graph reduction, done heuristically (none re-implemented the Independent_Set part), is shown to bring advantage to their approaches both in solution quality and in computation time. However, none of them investigated in-depth this approach which is only suggested at the end of their articles, with results given only on one or maximally two graphs.

It would be interesting, instead, to address some issues arising from this approach

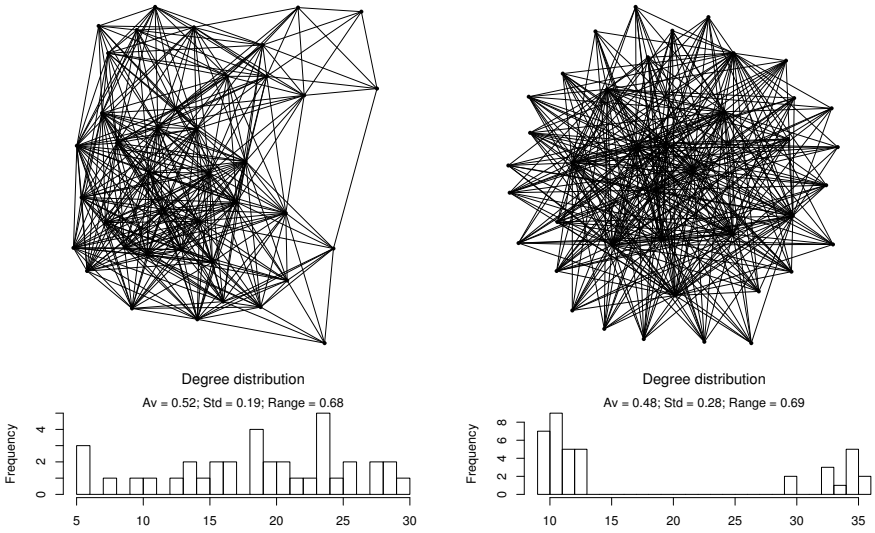


Figure 4.11.: Visual comparison of the structure of two graphs with similar characteristics but different hardness when solved by Independent_Set. The graphs are drawn using the GEM algorithm of Frick et al. (1994), which attempts to achieve pleasant layouts according to the most influential aesthetic criteria, like maintaining close connected vertices and avoiding crossing edges.

when using approximate algorithms. For example, it is not clear which should be the value to give to `EXACTLIM` since, through the introduction of approximate algorithms, it could be increased very much. More importantly, it is not clear whether the strategy of removing independent sets by leaving reduced graphs of lower edge density is a good strategy for the successive approximate algorithm or whether the opposite strategy would be more appropriate. In any case, the choice of a "strategy" in doing this may be the "right" one for some graphs but the "wrong" one for others. In this latter case, the reduced graph can even be coloured exactly but the resulting partition of the starting graph into sets can be far from that required by an optimal colouring. A remedy could be to embed some restarting mechanism such that a new graph reduction, different from the previous one is attained. We attempted to answer these questions in some preliminary experiments in which we combined $\mathsf{TS}_{\mathcal{N}_1}$ with the parametrised RLF with Independent_Set. We were, however, unable to devise any competitive algorithm to answer the posed questions and, hence, we decided to abandon this direction.

4.10.4. Implementation Details

All algorithms were written in C++ using the framework organisation suggested by Di Gaspero and Schaerf (2003a). There are two standard ways to represent a graph $G = (V, E)$: as a collection of adjacency lists or as an adjacency matrix. The adjacency list provides a more compact way to represent sparse graphs. Provided that we had enough main memory for the instances under examination (we work on a 1GB RAM machine), we decided to maintain both data structures because the lists indicate quickly which are the adjacent vertices for a given vertex and the matrix tells quickly if there is an edge connecting two vertices.

Also for the colouring we maintain two representations: a vector of colours, indicating

for each vertex its colour; and a collection of sets of vertices, one for each colour. The first derives from the mapping representation and the second from the partitioning representation. Given the high number of operations of insertion and removal from sets required by Stochastic Local Search algorithms and the fact that they occur in similar amount, we implemented each set in the collection as a binary search tree. In binary search trees, both insertion and removal require $\mathcal{O}(\log n)$ while a simple unordered list would require constant time for the insertion but linear time $\mathcal{O}(n)$ for the removal. The saving of computation time in our context is significant.

4.11. Experimental analysis on benchmark instances

In this section we describe the details of the experimental analysis conducted for the evaluation of the approximate algorithms. We recall that all these algorithms are stochastic procedures and many published computational comparisons are inadequate because they are based on best results. The best (or extreme) value of a distribution of stochastic results is known from Statistics to be a biased location estimator, *i.e.*, given the best of a sample, the best of the population from which the sample is drawn may be different. The comparison we present here is the first rigorous experimental analysis by means of statistical methodologies for the GCP.

In the experiment, we consider algorithms as levels of a treatment factor and instances as levels of a blocking factor. Each algorithm was run 10 times on each instance and the smallest number of colours for which a feasible colouring was found in each run is the response variable. The design reflects the *"several runs on various instances"* introduced in Section 3.6.6, page 58.

For each run the termination criterion was a fixed CPU time. The amount of CPU time is a crucial choice because it can bias the results of the evaluation and it depends on the specific application for which graph colouring is used. We decided to adopt an intermediate amount of time between very short run times required by real time applications (like memory allocation) and very long run time allowed by design applications (like some cases of timetabling). The scenario, with which we are concerned, is the *planning scenario* (see also Section 3.6) in which an answer must be obtained in a time no longer than a few quarters of an hour. Beside this typology of application, the amount of time was also selected in such a way that all algorithms in the comparison could be effective and exploit most of their potential.

After preliminary experiments, we decided to use $\mathsf{TS}_{\mathcal{N}_1}$ as the reference algorithm and to set as time limit the time it needs to perform $I_{max} = 10^4 \times |V|$ iterations.

The time limit. The value for I_{max} is necessarily an arbitrary choice. In principle, SLS algorithms could find further improvements in solution quality at any time and no precise procedure is known to predict when an algorithm becomes ineffective, that is, when the probability of finding improvements becomes sufficiently small to be negligible.

Our choice is a compromise between two criteria: achieving high quality results and limiting the use of computational resources. To support our decision we relied on observations of the empirical attainment curves or qualified run time distributions. In particular, we ascertained that the I_{max} iterations are enough for $\mathsf{TS}_{\mathcal{N}_1}$ to reach a median limiting behaviour. More precisely, I_{max} is selected such that after that threshold no further improvement in the median attainment curve of $\mathsf{TS}_{\mathcal{N}_1}$ is observed. This behaviour is shown

for two sample instances in Figure 4.12. The dashed vertical line represents I_{max} while the runs extend to, respectively 10 and 2 times I_{max}. It is evident that after I_{max} iterations no further improvement appears in the median or in the best case, although on instance DSJC1000.5 the last quantile attainment curve improves long after I_{max}. In general terms, for all instances of the DIMACS benchmark suite improvements for $\mathsf{TS}_{\mathcal{N}_1}$ after I_{max} are still possible but become very rare.

The need, due to the diversity of the algorithms, for transforming the limit expressed in number of iterations into a limit expressed in computation time, poses another problem. Being $\mathsf{TS}_{\mathcal{N}_1}$ a stochastic algorithm, the time needed to accomplish I_{max} iterations is also a stochastic variable. Hence, we replicated its observation 10 times per instance. Unfortunately, especially for the largest instances, the variance in the time required by Tabu Search is considerable. The explanation is in the variable size of the neighbourhood explored, defined by V_c, and in the value of the final k. However, the replication of the measure and the use of the median value should equitably distribute errors so that the reproducibility of the experiment is maintained.

In Figure 4.13, we give an indication of how the computation times of $\mathsf{TS}_{\mathcal{N}_1}$ vary with the size, the density and the number of iterations for random graphs.

Performance measurement. Each result $\widehat{\chi}$ on a graph G is transformed into

$$err(\widehat{\chi}, G) = \frac{\widehat{\chi}(G) - \widehat{\chi}^{best}(G)}{\widetilde{\chi}^{ROS}(G) - \widehat{\chi}^{best}(G)} \tag{4.2}$$

where $\widetilde{\chi}^{ROS}(G)$ is the median result attained by 10 runs of the ROS heuristic on G, and $\widehat{\chi}^{best}(G)$ is the best known solution for G (see also Equation 3.1, page 41).

In Figure 4.14, we report the box-plots for the distributions of $err(\widehat{\chi}, G)$. We note that it is very hard to distinguish where differences arise. The only few considerations arising are that (i) on the Full Insertion and the Insertion graphs all algorithms perform the same; (ii) on all classes of instances, approximate algorithms improve strongly over the ROS heuristic.

A deeper insight into the Full Insertions and the Insertions class of graphs revealed that for the Full Insertions graphs the optimal solution can be obtained in reasonable time (less than one hour) by exhaustive search if the preprocessing rules of Section 4.5 are first applied to reduce the graph. The optimal solution is however always found also by the RLF construction heuristic. Hence, these graphs are in fact easy. In contrast, some Insertions graphs, appear very hard because the best known solution given by Caramia and Dell'Olmo (2002c), if confirmed[14] is never reached.

With concern to the result on real life graphs, Coudert (1997) points out that it is very hard to find graphs coming from real life applications which have $\omega(G) \neq \chi(G)$. It appears, then, that clique values close to the chromatic number also help SLS algorithms.

Transforming the data within each instance into ranks, in such a way that each result is mapped into a value in $[0, 90]$, we start distinguishing some pattern. We then aggregated the ranks relative to instances of the same class and the resulting box-plots are shown in Figure 4.15. However, the differences which arise from the box-plots should be tested for significance and therefore we proceed with the statistical analysis of results.

[14] Caramia and Dell'Olmo (2002c) report that the optimal solution for the instance DSJC125.5 is 12 and it can be solved in much less than one hour, but it was impossible for us to verify this result, despite many efforts with very long runs of Ex-DSATUR or other algorithms introduced. The best result we found remains 17 colours.

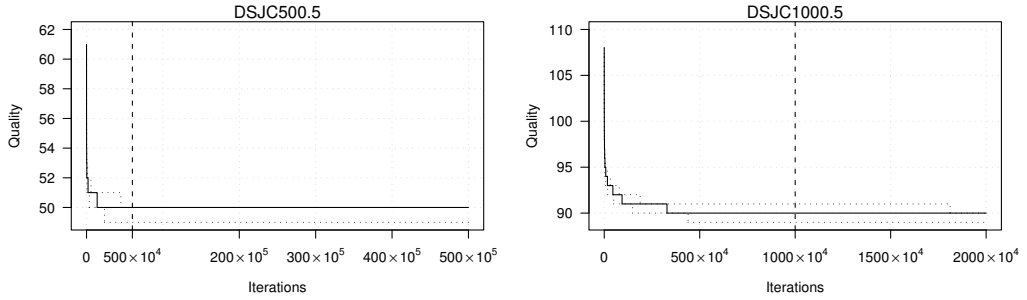


Figure 4.12.: Empirical attainment surfaces of $\mathsf{TS}_{\mathcal{N}_1}$ for 50 runs on `DSJC500.5` and 20 runs on `DSJC1000.5`. The empirical attainment curves represented are the median (continuous line) and the first and last quantile levels (dotted lines). The vertical dashed lines indicate the threshold of I_{max} iterations.

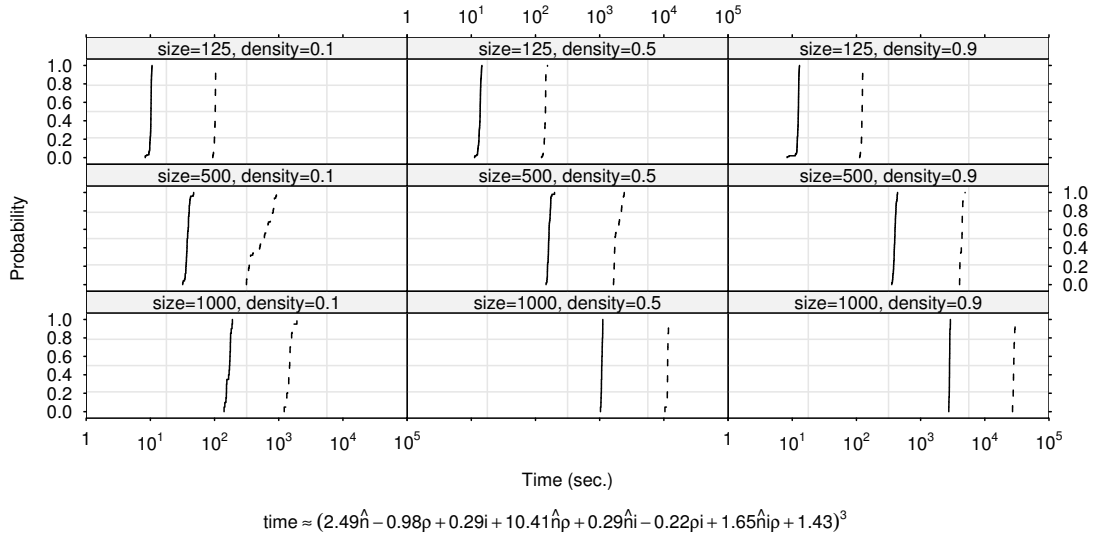


Figure 4.13.: The variation of the computation time to perform $1 \times I_{max}$ (continuous line) and $10 \times I_{max}$ (dotted lines) iterations on random graphs of different sizes (rows) and edge densities (columns). Represented are the empirical cumulative distribution functions of the stopping time. Data are obtained by 50 runs (20 in the case of size 1000 and density $\{0.5, 0.9\}$) per instance. The x-axis is in logarithmic scale. Clearly, time increases with size, edge density and I_{max}. The linear regression model in the subtitle yields $R^2 = 0.9959$. In the model, $\hat{n}$ is the number of vertices in the graphs divided by 1000, ρ is the edge density and i the multiplying factor for I_{max}.

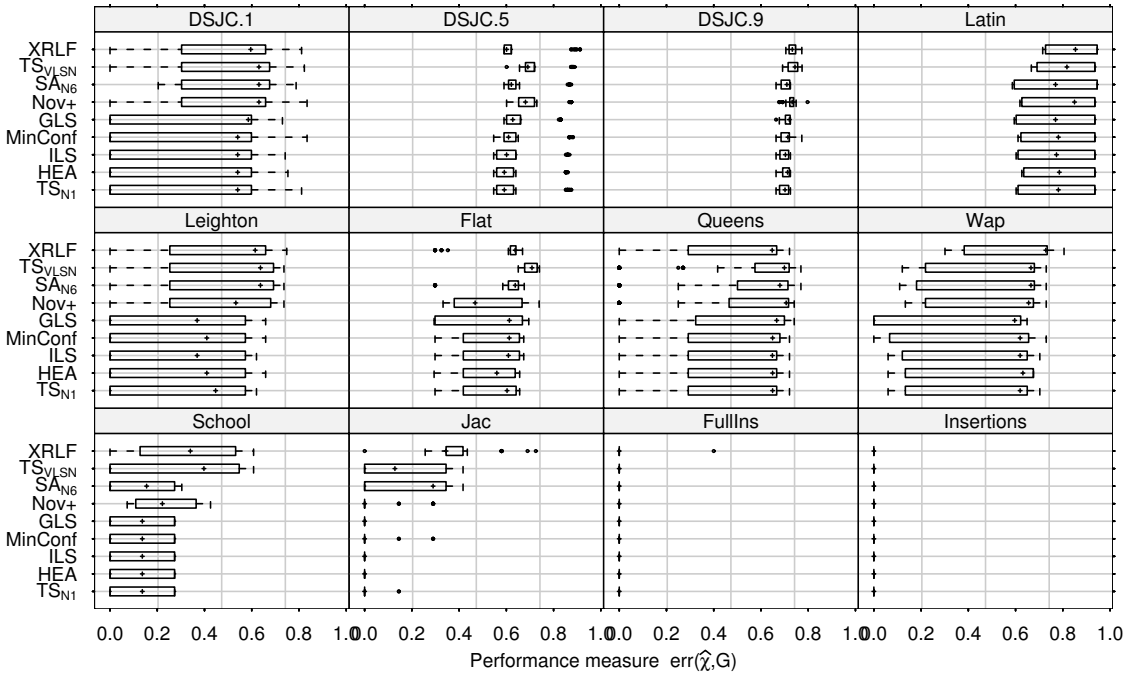


Figure 4.14.: Box-plots of the performance measure $err(k, i)$.

Application of statistical tests. We first remove from the analysis the classes of instances where no difference among the algorithms are found, *i.e.*, the Insertions and Full Insertions graphs. We treat the instances as a blocking factor and the class as a categorical stratification variable of the instances. To decide upon the supposed presence of an interaction between algorithms and classes we consider the interaction plot of Figure 4.16. Lines are not perfectly parallel and suggest that an interaction may be present. Yet, the plot does not tell whether the interaction is statistically significant. Precautionally, we include the interaction term in the linear model that represents the whole experimental design:

$$\text{response} = \text{constant} + \begin{matrix}\text{effect of}\\ \text{blocks}\end{matrix} + \begin{matrix}\text{effect of}\\ \text{algorithms}\end{matrix} + \begin{matrix}\text{effect of}\\ \text{classes}\end{matrix} + \begin{matrix}\text{effect of interaction}\\ \text{algorithms-classes}\end{matrix}$$

where we omit the error terms for the sake of conciseness. In order to state whether the assumptions for a parametric analysis are met, we consider the standardised residuals of the fitted linear model. In Figure 4.17 we try to determine visually whether errors are independent, homoschedastic, and normally distributed. The plot on the left shows that standard residuals are not homoschedastic. Moreover, the normal plot on the right, shows that residuals deviate, although not excessively, from the normal distribution.

A transformation of data may help to get closer to meet the parametric assumptions. Venables and Ripley (2002) point out a method to select automatically the best transformation. The method determines the best λ and α value for the following family of transformations:

$$y^{\lambda} = \begin{cases} (y^{\lambda} - 1)/\lambda & \lambda \neq 0 \\ \log y & \lambda = 0 \end{cases} \qquad \text{and} \qquad t(y, \alpha) = \log(y + \alpha).$$

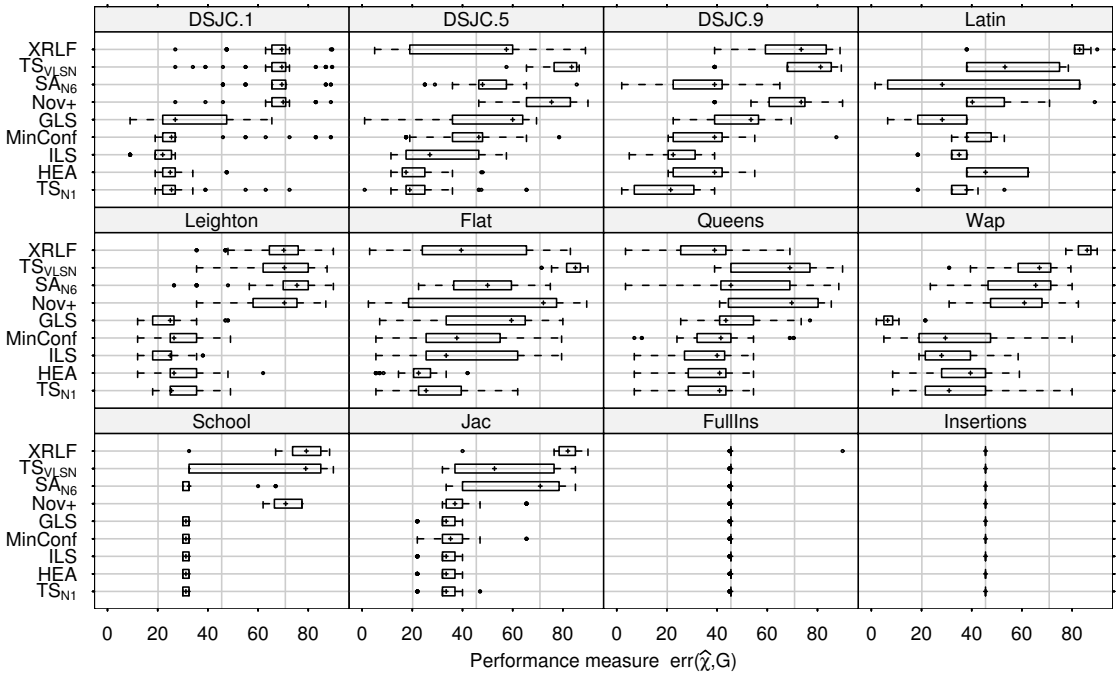


Figure 4.15.: Box-plots of the experimental results ranked on the instances.

where y are the observed responses. The diagnostic plots for the model with the transformed data in Figure 4.18 shows that the assumption of the normality of data is still not satisfied.

We consider, therefore, non-parametric analyses. In this case, we simplify the analysis studying separately the results on each instance class. This simplification also avoids biases due to the different number of instances available on each class. In Figure 4.19, we report the simultaneous confidence intervals as obtained by Tukey's HSD method (Equation 3.15, page 59), by CSP synchronised permutations (see Algorithm 3.6, page 63), and by the rank test of Friedman (Equation 3.19, page 62). In addition to this, we report in Figure 4.20 the results of the graphical representation of Hsu (1996) which permits to gain statistical power in the comparisons by allowing the confidence intervals to assume different widths (this may be seen as a workaround for the violation of the assumption of homoschedasticity of the results among the algorithms).

We first discuss the relationships among the different statistical tests, since the results are sometimes contradicting. Then, in the next paragraph, we discuss the conclusions for the comparison of the algorithms.

The first observation is that the three test methodologies produce in some cases a different inference. A significant difference arises on the class of the Flat graphs where Nov$^+$ receives a much better evaluation with permutation and parametric tests than with the rank-based test. This result is due to the fact that with the rank transformation the distance between the colourings found by algorithms is lost (on graph `flat1000_60_0`, Nov$^+$ finds solutions with 64-65 colours while all other algorithms do not go below 87 colours; this fact has a strong influence on the average error measure). Other differences concern the presence or not of statistical significance in the comparisons.

In general, permutation tests result less powerful than the other two methods because

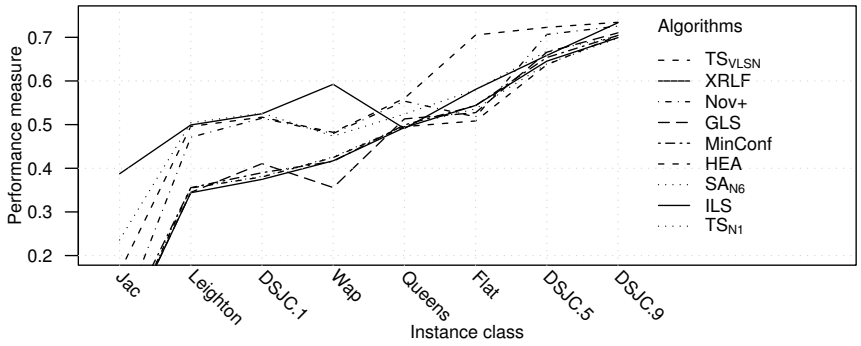


Figure 4.16.: The effect of interaction between algorithms and instance classes on the median value of the error measure.

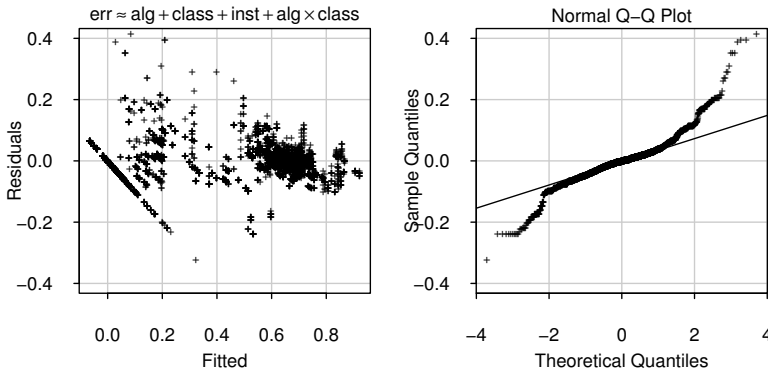


Figure 4.17.: Diagnostic plots for checking the assumptions of parametric ANOVA. Despite an $R^2 = 0.9583$ in the linear model, the assumptions of normality are not met, as mainly seen from the strong non-normal tails in the plot on the right.

the intervals are always larger. This may indicate that Algorithm 3.6 is too conservative and, indeed, this is also the outcome of the study in the Appendix B. A further explanation may also be that the assumption of homoschedasticity is violated. Since Algorithm 3.6 searches for the confidence intervals that satisfy all the comparisons, the algorithms with highest variance bias negatively the whole analysis (we recall that homoschedasticity is assumed by all three test methodologies). The representation of Figure 4.20 allows permutation tests to become the most powerful. In Appendix B, we show that, for homoschedastic distributions, the procedure for obtaining the plots of Figure 4.20 has the same type I error rate as the other methods of Figure 4.19. Comparing the permutation tests in the two representations we observe that larger confidence intervals are due to few algorithms whose performance are not among the best. However, since both permutation and rank-based tests are based on the same assumptions, we base the following final comments on the rank-based results of Figure 4.19. The choice about which analysis procedure is more appropriate depends ultimately on the final application of the algorithm and hence on the real context. If there was a cost assigned to each colour then probably it would be better to rely on the inference produced by permutation tests. Our choice for the rank-based test is due to the fact that we do not have a cost associated to the number of colours and, typically, the interest in the graph colouring literature is rather on which algorithm "wins" more frequently.

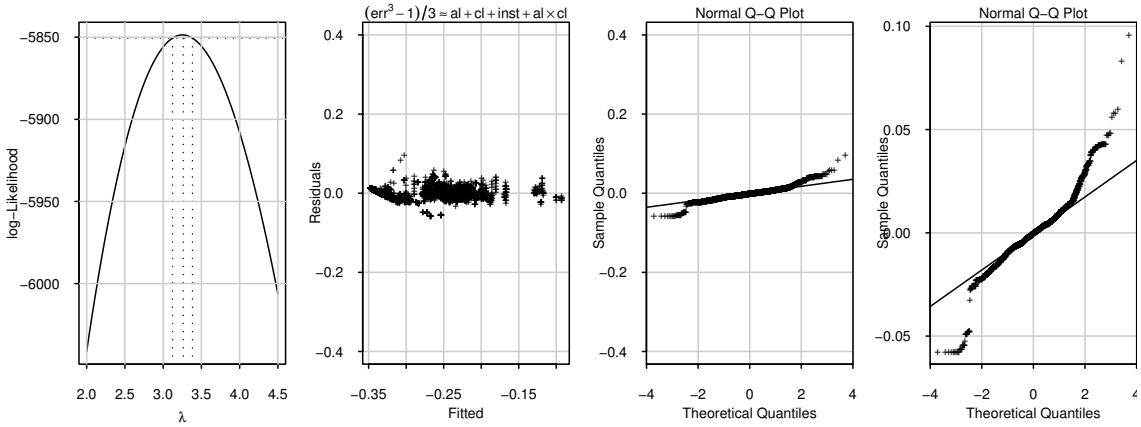


Figure 4.18.: The effect of data transformations on the linear model. The first plot on the left shows the linear model likelihood function which is maximised for a λ value close to 3 (vertical lines show an approximate 95% confidence interval for good values of λ). The next three plots are diagnostic plots for the model with the transformed data. The model yields $R^2 = 0.947$ and seems to improve both the spread of residuals and the normality of data, if the scales of plots are maintained as in Figure 4.17. However, in a different scale (fourth plot on the right) data are far from approximating the normal distribution.

Conclusions for the comparison. The following conclusive statements can be made.

- There are differences in the relative order of the algorithms through the classes of instances; therefore, it is not possible to declare one method as the only best one.
- On the random graphs $\mathsf{TS}_{\mathcal{N}_1}$, ILS, and HEA are the most competitive algorithms. On these graphs, HEA's performance worsens with respect to the other algorithms when density increases. $\mathsf{TS}_{\mathcal{N}_1}$ is significantly worse than HEA only on graphs of density 0.1 while it is significantly better for density 0.9. Worth to note is that ILS performs significantly better than $\mathsf{TS}_{\mathcal{N}_1}$ only on the density 0.1 graphs. This better performance is the minimal requirement for these two algorithms to make a contribution since both are constructed upon $\mathsf{TS}_{\mathcal{N}_1}$.
- On the Flat graphs, HEA is the best algorithm, although its results on the instances of size 1000 vary considerably (see numerical data in the Appendix C.1.4).
- On the Leighton graphs, ILS and GLS are the best algorithms and they do not differ significantly.
- The most evident difference in performance through the classes is the one of GLS which is the best algorithm for the WAP class and the Leighton class, while it is worse than other algorithms in almost all the other classes.
- On the Jacobian Estimation graphs, there are 6 algorithms with almost the same performance.
- $\mathsf{TS}_{\mathsf{VLSN}}$ is never competitive in any class of graphs; rather, it ranks always among the worst algorithms.
- On all classes, XRLF performs poorly. It is competitive only on the Queens graphs where it achieves the best median results, although its performance are not significantly better than those of ILS and HEA. XRLF performs poorly also on the random graphs DSJC.5 although, as reported in Appendix C.1.4, it solves the instance

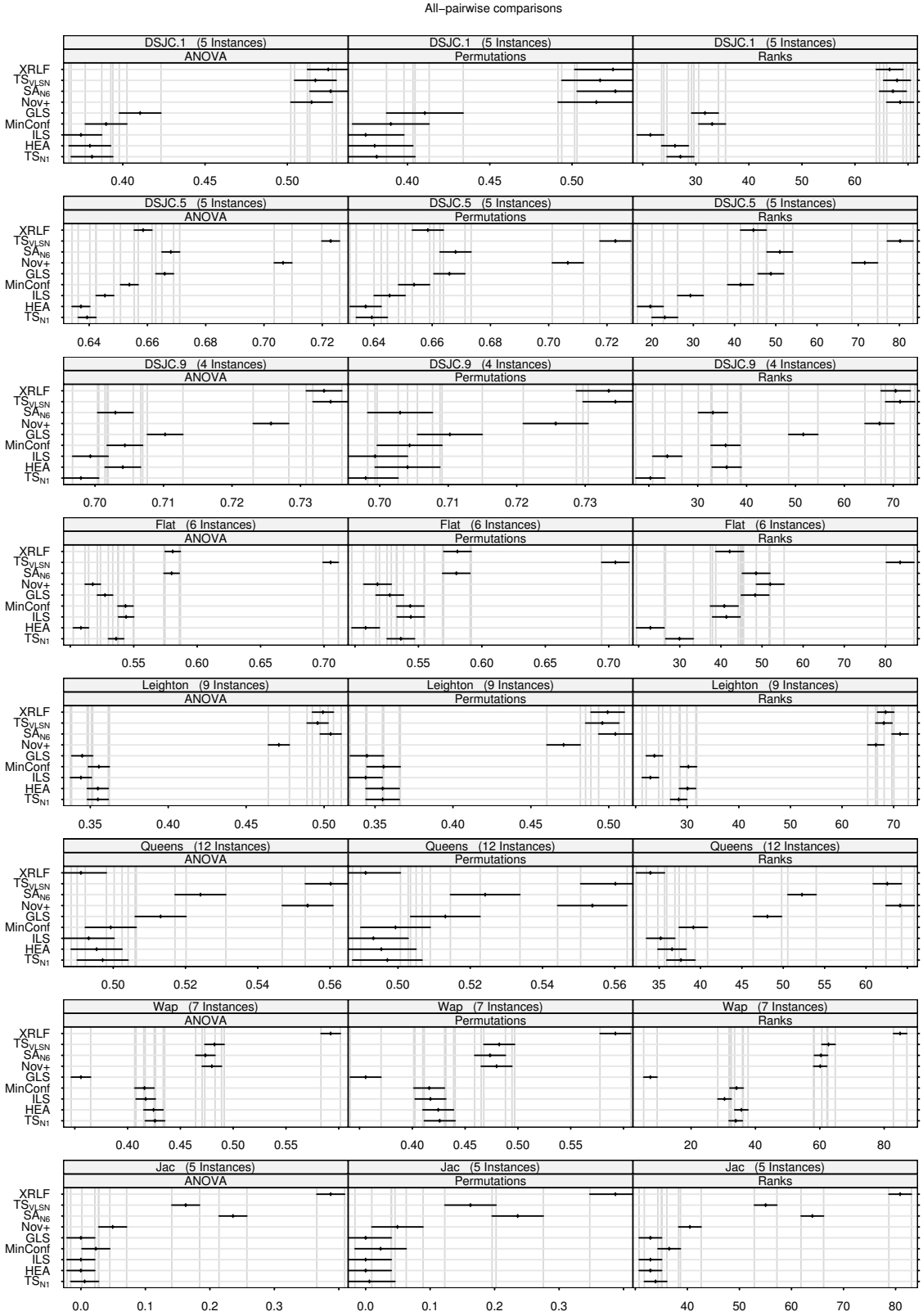


Figure 4.19.: Confidence intervals for the all pairwise comparisons of approximate algorithms for the GCP. The confidence intervals for three test methodologies are given: parametric based on average error measure (left column), permutation based on average error measure (central column), and rank based on average rank (right column).

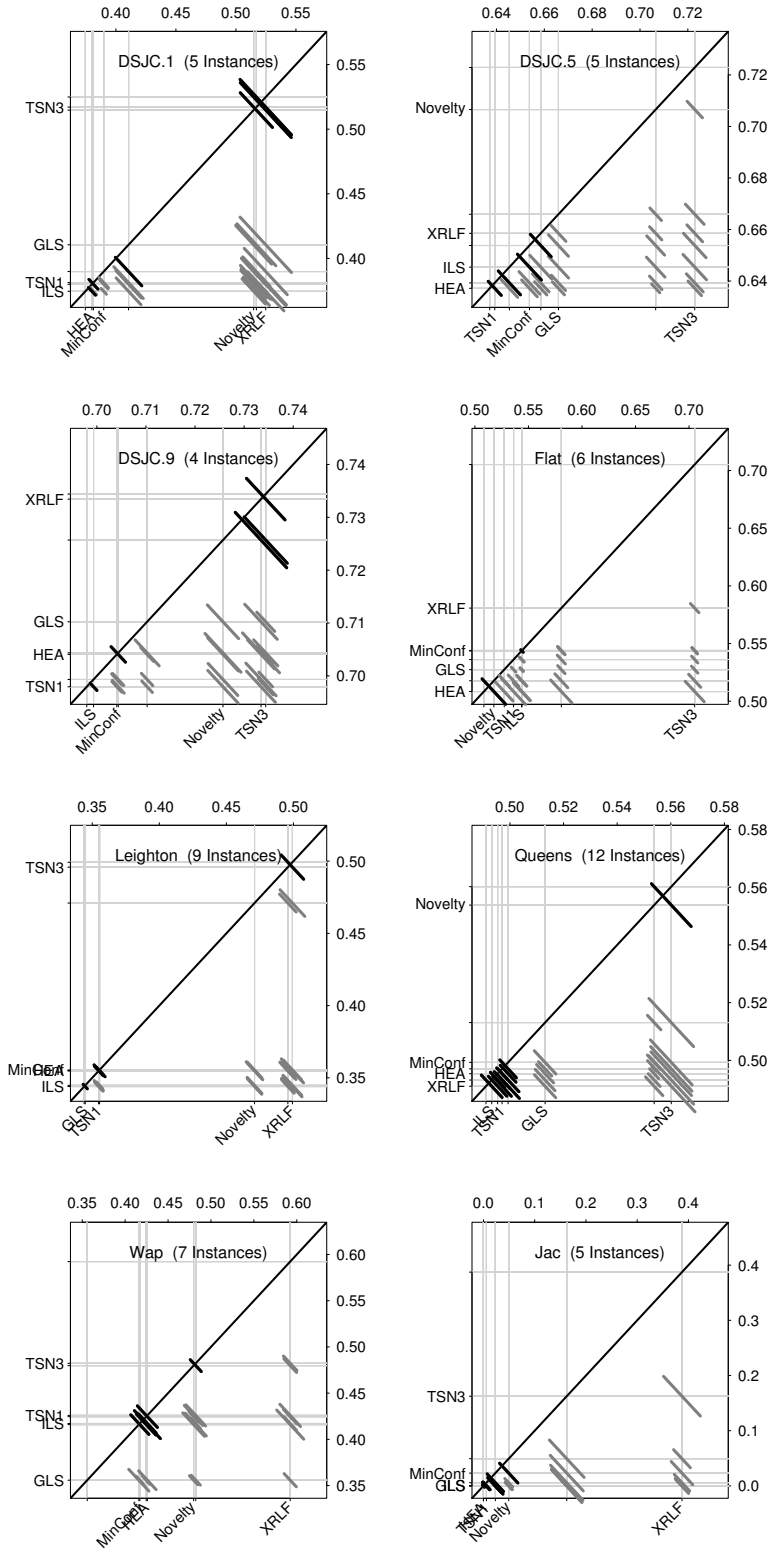


Figure 4.20.: Confidence intervals for permutation tests in the all-pairwise comparison procedure with the graphical representation introduced in Section 3.8, page 66. The names of the algorithms are alternated between the x- and y-axis. The best algorithms are those whose names are close to the origin of the graph. Statistical inference on differences is carried out by checking whether a segment representing the minimal significant difference between two algorithms crosses the $-45°$ line. Intervals that do not entail statistical significance are represented in grey.

DSJC.1000.5 with 86 colours which is the best result on that instance. We put emphasis on this result because many comments in favour of this algorithm appeared in the literature are based mainly on its performance on this instance.

We conclude with a final comparison with exhaustive search. Branch and bound (ExDSATUR) on the instance DSJC1000.5 finds an initial 115-colouring but does not improve it in the next 1102 seconds which is the time limit for that instance. All the approximate methods in our study find, by a large margin, a better colouring. Also the incomplete randomised backtracking method with forward checking proposed by Prestwich (2002a) performs poorly on this instance, yielding a best result of 97 colours, which is still far from our results.

All numerical results for the individual instances and comparisons with benchmark results are reported in Appendix C.1.4.

4.12. Further analyses

4.12.1. On the time dependent profile

So far, we compared solution quality on a fixed time limit. Nevertheless, as described in Section 3.8.1, page 69, a more thorough analysis of performance should take into account also run time.

In Figure 4.21, we plot the median empirical attainment curves of the solution quality over run time for some meaningful cases. In many of the other instances the profiles are mostly overlapping and, therefore, not indicative of any significant differences in algorithm behaviour. In the instances presented, the final colouring found by all algorithms is never optimal; hence, the curves are not bounded from below and a "flat" line indicates that the algorithm is not able to find a better solution in the available computation time. The time is normalised on the maximal time allowed for the specific instance (see Appendix C.1.4 for the precise value of the time limit). We can draw the following conclusions.

- In general, the chosen time limit allows algorithms to exploit their potential and be effective. Curves tend indeed to become "flat" towards the end of the run, indicating that further improvements in solution quality are unlikely to happen.

- HEA is an exception. Its curve does not reach a "flat" trend and the time limit appears to be too restrictive for this algorithm above all on the large random graphs. To test this conjecture, we run HEA on instance DSJC1000.5 with the best set of parameters suggested by the authors (Galinier and Hao, 1999). We report the results in Table 4.7. We note that in the long term HEA is able to improve considerably the results, finding an 84-colouring. $\mathsf{TS}_{\mathcal{N}_1}$, in contrast, is not able to go beyond an 88-colouring for this instance even allowing the same time as HEA. These results confirmed those presented by Galinier and Hao (1999) when parameters are properly tuned.

  In our specific experimental setting we chose to run HEA with a fixed number of 10000 iterations of the embedded Tabu Search, while in the original paper, it was

DSJC1000.5

k	success	time (sec.)	tot. iterations	par. iterations	crossover	restart
92	5	461.69	1356000	406973	19	0
91	5	731.17	2109509	753509	58	0
90	5	1240.73	3536215	1426706	71	0
89	5	1964.33	5607288	2071073	120	0
88	5	2725.87	8007345	2400057	142	0
87	5	4066.91	11935038	3927693	223	0
86	5	6428.91	18625357	6690319	342	0
85	5	8725.42	27417444	8792087	491	0
84	4	14998.58	46398132	18980688	1338	1
83	0	36000.00	116221109	69822977	4525	3.5

Table 4.7.: Detailed results for HEA when the run time on the instance DSJC1000.5 was fixed to 10 hours. (In our experimental design, the maximal time allowed for this instance was 1102 seconds.) The number of iterations of $\mathsf{TS}_{\mathcal{N}_1}$ before applying the GPX crossover was set to 16000 while the parameter δ was set to 0.6. The algorithm was run 5 times and almost all runs produced an 84-colouring while they failed to solve the 83 case. Reported are the median time for producing each colouring, the total number of iterations, the number of iterations required to solve only the specific colouring, the median number of crossover recombinations, and, finally, the number of restarts at the current best k. Results are comparable with those of the original paper on HEA.

varied in relation to the actual value of k and reached 16000 iterations for the smallest values of k. Our choice was a compromise between the need to use a large number of iterations which allows $\mathsf{TS}_{\mathcal{N}_1}$ to solve rapidly instances if k is large and the need to test an algorithm in which the recombination really plays an effective role within the given time limit. Moreover, we gave preference to robustness, that is, not to have to change parameters for each k and for each instance.

We add that a similar long time experiment on the instance DSJC1000.5 was performed for ILS but results showed that it is not even able to outperform $\mathsf{TS}_{\mathcal{N}_1}$.

- $\mathsf{TS}_{\mathcal{N}_1}$ is the best choice if the available time is small. Indeed, no other algorithm can do better in the first 20% of the time.

- On the instance flat1000_50_0, as also on the instance flat1000_60_0 not reported in Figure 4.21, the behaviour of the algorithms is anomalous. All algorithms encounter difficulties to solve colourings with about 90 colours which is much more than the chromatic number for those graphs. The only algorithm which succeeded in all 10 runs to find a colouring with less than 56 colours is Nov^+. $\mathsf{TS}_{\mathcal{N}_1}$ reached 50 colours in one single run but in all the others it could not go below 86 colours. In general, for these graphs, once a range of colours between 100 and 86 is passed, Nov^+ as well as other algorithms proceeded very fast to less than 60 colours. This strange behaviour was already observed by Culberson and Luo (1996), who identify ridges of resistance in terms of the number of colours, where the number of iterations needed by their iterated greedy algorithm to find a feasible colouring grows exponentially as the known chromatic number of the graph increases. This is the case for both, Uniform and Flat random graphs, although Flat graphs are harder to solve. Culberson's analysis entails that the graph flat_1000_76 is even harder to solve, in the sense that it requires more iterations for passing the region where the ridge of resistance arises. This is confirmed by our experiments in which no algorithm was able to exit in the allowed time from the region of resistance localised approximately between 101 and 90 colours. For these graphs therefore the time limit should be reconsidered. This result, however, is important and must be taken

into account when comparing algorithms for the chromatic number problem. Solving only the decision version of the problem at the known chromatic number can be misleading, since this can be much easier than trying to approximate this value by solving a series of decision problems. For example on the instance `flat_1000_50` all algorithms of our analysis found a 50-colouring if started from less than 60 colours!

- $\mathsf{TS}_{\mathsf{VLSN}}$ is quite slow. Furthermore, no evidence arises that longer running time could make this approach competitive.
- On the instance `le450_15d`, we were unable to tune XRLF in order to produce a solution within the time limit and its first result arrived after the time limit expired.

4.12.2. On the Tabu Search in the very large scale neighbourhood

The main reason for the failure of the very large scale neighbourhood to perform better than $\mathsf{TS}_{\mathcal{N}_1}$ is the high computational cost of the neighbourhood exploration. Figure 4.22 compares the behaviour of $\mathsf{TS}_{\mathcal{N}_1}$ and $\mathsf{TS}_{\mathsf{VLSN}}$ under two different perspectives: over time and over number of iterations. It can be observed that $\mathsf{TS}_{\mathsf{VLSN}}$ remains competitive comparing results based on iterations but each iteration with VLSN costs much more than an iteration using only $\mathcal{N}_1$. Hence, the total number of iterations performed in the same time with $\mathsf{TS}_{\mathsf{VLSN}}$ is much less than with $\mathsf{TS}_{\mathcal{N}_1}$. For the same time, the loss in the number of iterations should be balanced by a higher solution quality. This appears, indeed, to be the case, as the curve of $\mathsf{TS}_{\mathsf{VLSN}}$ is below the one of $\mathsf{TS}_{\mathcal{N}_1}$ when comparing the number of iterations, although the gain is small. In other terms, infeasible colourings which cannot be improved by an exchange in $\mathcal{N}_1$ but only by using a VLSN exchange are not many and in any case not so many to justify the use of VLSN. This confirms the results discussed in Section 4.9 and leads us to conjecture that the search space of graph colouring has many well spread local optima and hence intensifying the search around them is not useful.

4.12.3. On the Tabu Search in the one-exchange neighbourhood

The main result of the analysis is that $\mathsf{TS}_{\mathcal{N}_1}$ remains very competitive on all the instances classes for the given time limit. However, further experiments also revealed that in some graphs like `DSJC1000.5` it was not able to go beyond 88 colours despite much longer computation time while we know that for that graph the best known solution is 83. This may suggest that after long run time Tabu Search becomes ineffective and the exploration remains confined to a restricted area of the search space.

An analytical study on the search landscape of Tabu Search for the k-colouring problem has been attempted by Hertz et al. (1994). The study is interesting and the computations presented may be helpful for estimating the optimal size of the tabu list or the number of iterations without improvement which should be done before stopping the search. However, indications are restricted to graphs of limited size. The landscape seems to change considerably according to k and the application of the same computations to large graphs becomes early infeasible.

We attempted here to detect through exploratory data analysis, whether Tabu Search tends to visit repeated solutions for long run times. For each vertex, we counted the

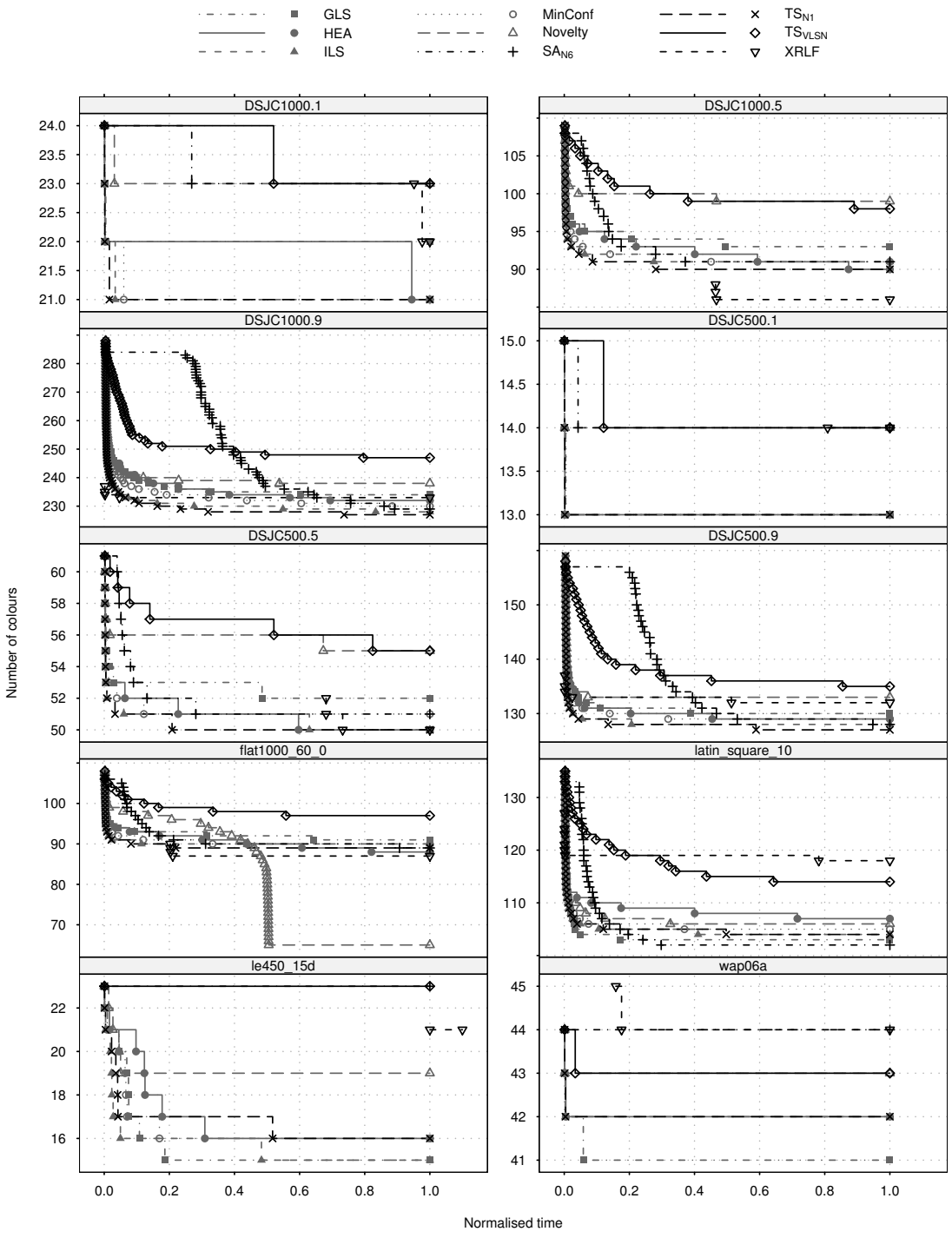


Figure 4.21.: Empirical median attainment curves for a unified representation of run time and solution quality performance. Given are 10 representative instances. On the x-axis, times are normalised by the maximal time allowed to solve the specific instance.

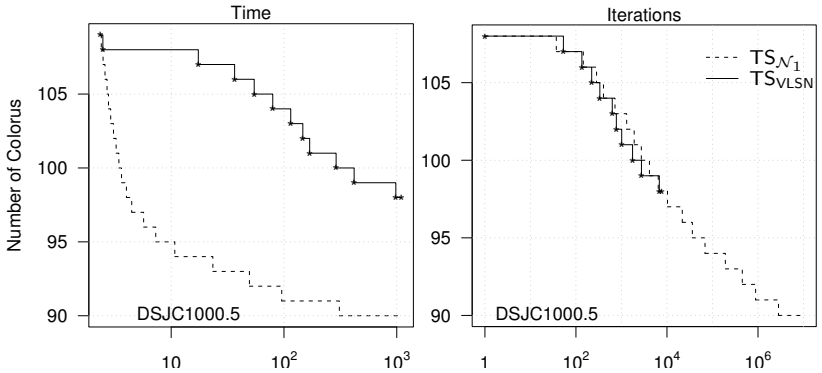


Figure 4.22.: The development of Tabu Search on $\mathcal{N}_1$ and VLSN over time and number of iterations. Represented are the empirical attainment curves across 10 runs per algorithm.

number of times its colour is changed and the number of different colours assigned. In Figure 4.23 we use Pareto charts to visualise the results on different instances and number of colours solved. In each Pareto chart, vertices are represented on the abscissa and are ranked from most "changed" to least "changed". Vertical bars represent the number of different colours assigned to each single vertex, while the line superimposed over the bars indicates the cumulative percentage of iterations in which a colour change occurred for the corresponding vertex. For each instance, four colours are represented, among them the best k for which a feasible solution was found in that run and $k-1$. The number of iterations was set to $2 \times I_{max}$.

The empirical data show that, apparently, colour changes are equally distributed among the vertices for the value of k where most of the iterations are needed. Furthermore, the number of different colours received by each vertex approximates the total number of available colours. Far away to be conclusive, these observations are in favour of the conjecture that the degree of search diversification in $TS_{\mathcal{N}_1}$ is high enough and that the algorithm does not cycle among visited solutions.

4.13. Experimental analysis on a large set of random graphs

The previous experimental analysis was limited to instances from the DIMACS repository and the number of instances per class was quite low. The result proved by Birattari (2004a) and discussed in Section 3.6, page 46, suggests instead that the best design for an experiment is *"one single run on various instances"*. Moreover, very little is actually known about the structure of some DIMACS instances and it is hard to link the performance of the algorithms to some general features of the graphs.

There are some graph generators available on the Internet for generating specific graphs. However, for many of them the maximal size of the graphs is typically limited by the combinatorial explosion of the generation process. Culberson's publically available random generator allows instead to create families of test graphs of large size that vary in terms of edge density and structure. The generator also provides a good upper bound (*i.e.*, a k for which it is known that a feasible k-colouring exists) and, therefore, offers the possibility to learn more about the performance of algorithms and their relations to graph properties.

Structure in a graph may be induced by limiting the clique size or by embedding cy-

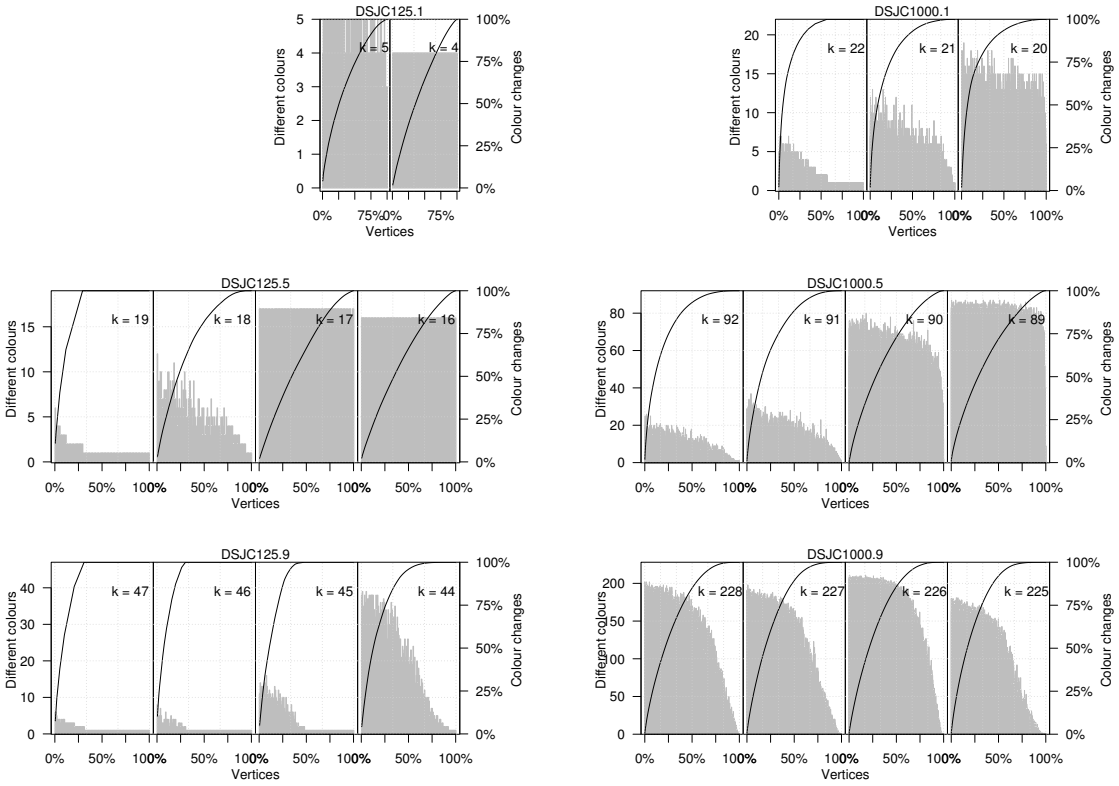


Figure 4.23.: Pareto charts for 6 runs of $\mathsf{TS}_{\mathcal{N}_1}$ on random instances of different size and density. Vertices are represented on the x-axis and are ranked by the number of colour changes. Vertical bars represent the number of different colours received, reported on the left y-axis, while the superimposed line indicates the cumulative percentage of colour changes, reported on the right y-axis.

cles larger than a certain length. It is also possible to impose a graph to have at least a given number of independent sets and to influence the variability of the size of these independent sets. The process of assuring a number of independent sets corresponds to hiding (or pre-specifying) a colouring number in the graph, which then becomes an upper bound to the chromatic number of the graph. The generator accepts inputs to influence all these features. We distinguish three features in our analysis: the presence of a hidden colouring, the variability of the size of the hidden independent sets, and the graph type determined by the characterisation of the edge probability.

We used this generator to produce a large number of test graphs for our second experiment. In the following we describe the generated graphs.

Hidden colouring. We distinguish between graphs without embedded colouring and graphs with a guarantee on the upper bound to the chromatic number. For these latter graphs, the number of independent sets in the construction is decided in relation to other features of the graph such as size and density. The alternatives used for the generated graphs are summarised in Table 4.8.

Variability of the size of the hidden independent sets. We adopted the typology *smooth k-colourable* graphs to bias the variability of the size of the colour classes. A graph is generated by assigning each vertex to the independent set with label $\lfloor kx(ax + 1 - a) \rfloor$, where $a \in [0, 1]$ is a *variability parameter* and x is a random number from

the interval $[0, 1)$. For $a = 0$, the size of the independent sets tends to be nearly equal, while for $a = 1$ a certain variability is embedded in the graph. This procedure allows therefore to range from Equi-partite colouring graphs to highly variable colouring graphs by controlling just one parameter. We generated graphs using $a \in \{0, 1\}$. This gave us the possibility of describing synthetically, with one single measure the structural variability of the graph, which is deemed responsible for differences in the hardness of GCP instances (Culberson et al., 1995),

Type of graph. We considered the following three edge probability characterisations:

Uniform graphs (independent random edge assignment). An edge between a vertex pair u, v is included with a fixed probability. These graphs are denoted as G_{np} and G_{knp}, if a k-colouring is embedded.

Geometric Graphs. These graphs are created by uniformly generating n points in the two dimensional square given by $0 \leq x, y < 1$. Vertices in the graph correspond to the points in this square and an edge is associated to a pair of vertices if their Euclidean distance is less or equal a value r. We denote these graphs as U_{nr}. The generator allows to maintain embedded k-colourings also in geometric graphs; we denote such graphs U_{knr}. According to Johnson et al. (1991), geometric graphs have "inherent structure and clustering". For statistics on the vertex degree of geometric and uniform graphs we refer to Figure 4.2, page 96.

Weight Biased Graphs. Weighted graphs try to challenge algorithms that use the clique structure to aid in obtaining colourings. They are designed to restrict the development of larger cliques while possibly distributing edges as equitably as possible between the pairs of embedded independent sets. As a side effect, they might have the smallest variability in vertex degrees. Previous studies have shown that these graphs are among the hardest to solve (Culberson et al., 1995), although their statistics on vertex degree looks very similar to those of Uniform random graphs. The graphs are generated by first assigning vertices to independent sets. Then, a weight W is given to all $\binom{n}{2}$ vertex pairs, except those in the same hidden independent set, which are assigned a weight of 0. Vertex pairs are then selected as edges with probability proportional to their weight. When an edge is added to the graph weights are decreased in such a way that the formation of large cliques becomes less likely. This procedure is controlled by two parameters, α and γ, that, according to Culberson et al., 1995, we set to 0 and 1, respectively (more details are available in the manual of the generator[15]). The process terminates when either all weights are zero, or when $\lfloor p\binom{n}{2} \rfloor$ edges have been selected. The edge density of the resulting graph depends on the parameters p and W. In order to attain graphs of edge density $\{0.1, 0.5, 0.9\}$ we set p to values in $\{0.1, 0.5, 0.9\}$ and $W \in \{2, 114, 404\}$ and $W \in \{4, 230, 804\}$ for graphs of size 500 and 1000, respectively (for details on the choice of these values see Culberson et al., 1995). We denote these graphs with W_{np}, and with W_{knp} if a k hidden colouring is embedded.

In Table 4.8, we summarise the characteristics of the 1260 graphs that we generated. For each parameter setting we constructed 5 graphs. The control of the variability in the

[15] J. Culberson. "A Graph Generator for Various Classes of k-Colorable Graphs." June 2002. http://web.cs.ualberta.ca/~joe/Coloring/Generators/generate.html. (June 2005.)

Size	Type	Density	Variability	Hidden colouring	Tot. graphs
500	G	0.1	0	5 10	10
			1	5 10	10
			no	–	10
		0.5	0	20 30 40 50 60	25
			1	20 30 40 50 60	25
			no	–	10
		0.9	0	20 30 40 60 80 100 120 140	40
			1	20 30 40 60 80 100 120 140	40
			no	–	10
	U	0.1	0	10 20	10
			1	10 20	10
			no	–	10
		0.5	0	20 30 40 50 60 70 80 90 100 110	50
			1	20 30 40 50 60 70 80 90 100 110	50
			no	–	10
		0.9	0	100 150 200 250 300	25
			1	100 150 200 250 300	25
			no	–	10
	W	0.1	0	5 10	10
			1	5 10	10
			no	–	10
		0.5	0	20 30 40 50 60	25
			1	20 30 40 50 60	25
			no	–	10
		0.9	0	30 60 90 120	20
			1	30 60 90 120	20
			no	–	10
1000	G	0.1	0	5 10 20	15
			1	5 10 20	15
			no	–	10
		0.5	0	20 30 40 50 60 70 80 90 100 110 120 130 140	75
			1	20 30 40 50 60 70 80 90 100 110 120 130 140	75
			no	–	10
		0.9	0	20 50 100 150 200 250	30
			1	20 50 100 150 200 250	30
			no	–	10
	U	0.1	0	20 30 40 50	20
			1	20 30 40 50	20
			no	–	10
		0.5	0	20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200	95
			1	20 30 40 50 60 70 80 90 100 110 120 130 140 150 160 170 180 190 200	95
			no	–	10
		0.9	0	100 200 300 400 500 600	30
			1	100 200 300 400 500 600	30
			no	–	10
	W	0.1	0	10 20	10
			1	10 20	10
			no	–	10
		0.5	0	20 30 40 50 60 70 80 90	40
			1	20 30 40 50 60 70 80 90	40
			no	–	10
		0.9	0	30 90 150 220	20
			1	30 90 150 220	20
			no	–	10

Table 4.8.: All the graphs generated. In total, 1260 graphs were generated of which 520 are of size 500 and 740 of size 1000.

size of the independent sets is possible only when the colouring number is hidden. When no hidden colour is embedded, the variability of the independent sets is not controlled and we denote it with "no". Table 4.9 gives aggregate statistics on the number of graphs considered.

$n = 500$	$\rho = 0.1$	$\rho = 0.5$	$\rho = 0.9$
G	60	90	90
U	60	150	150
W	30	75	60

$n = 1000$	$\rho = 0.1$	$\rho = 0.5$	$\rho = 0.9$
G	90	180	180
U	120	150	180
W	30	120	60

Table 4.9.: The number of graphs per class given by size, edge density and graph typology.

Analysis of the termination criterion for $\mathsf{TS}_{\mathcal{N}_1}$

We first analyse the appropriateness of the termination condition for $\mathsf{TS}_{\mathcal{N}_1}$ adopted in the previous experiment which yield the time limits reported in Table 4.10. We take into

	$\rho = 0.1$	$\rho = 0.5$	$\rho = 0.9$
$n = 500$	60	120	180
$n = 1000$	155	465	720

Table 4.10.: The time limits in seconds for the instances differentiated by size (rows) and density (columns).

consideration two aspects: the maximal number of iterations and the computation time that $\mathsf{TS}_{\mathcal{N}_1}$ needs to perform this number of iterations.

In Figure 4.24, left, we plot the probability of finding a feasible colouring and an improvement in solution quality (*i.e.*, a decrease in the number of colour conflicts) after a certain number of iterations. The graph is obtained by recording the last corresponding events occurred in a run of $\mathsf{TS}_{\mathcal{N}_1}$ on each of the 740 random graphs of size 1000. $\mathsf{TS}_{\mathcal{N}_1}$ was run on all graphs for a maximal number of $100 \times I_{max}$ iterations. The curves correspond to the empirical survival distributions of the event "finding a colouring with less constraint violations" or "finding a feasible colouring". In other words, each point indicates the probability of finding a graph where improvements in the number of constraint violations or in the number of colours can still occur after a certain number of iterations, reported on the x-axis. Both kind of improvements after $50 \times I_{max}$ iterations occurred on only 10 graphs, that is, with a probability lower than 0.02. The number of colouring violations in these extreme cases was below 13 and, probably, further improvements could have been possible with longer runs. Therefore, the data should be treated more correctly as censored data and the curve be truncated. Nevertheless, for 730 of the 740 graphs no further improvement in the solution occurred in the last $50 \times I_{max}$ iterations, which lets us conjecture that further improvements will not occur on those graphs. However, the plot also indicates, that after I_{max} iterations a feasible colouring was still found for 26% of the graphs. In other terms, setting the termination criterion to I_{max} corresponds to miss 26% of cases where a better colouring can still be found. This empirical probability falls below 3% after $10 \times I_{max}$. The probability of finding improvements in solution quality is slightly higher, 35% after I_{max} and 5% after $10 \times I_{max}$. In the light of these results, $10 \times I_{max}$ would have been a better choice for the termination criterion of $\mathsf{TS}_{\mathcal{N}_1}$. This increase, however, corresponds on average, to an increase in computation time of a factor of 10.

In Figure 4.24, right, we investigate the computation time corresponding to I_{max} iterations on graphs of 1000 vertices. Clearly, the reason for uncommonly long run times for $\mathsf{TS}_{\mathcal{N}_1}$ to perform I_{max} iterations is a high number of violations in the solutions since the neighbourhood size of $\mathcal{N}_1$ is dynamically linked to the number of colours and the number of vertices involved in at least one colour conflict. The plot shows the correlation between the number of violations present in the final solution and the time elapsed between the time-point when a last feasible colouring is found and when the maximal number of iterations is passed. Evidently, long run times arise in cases where a lot of time is spent for numbers of colours that cause many violations. Very likely, in those cases there exists no better feasible colourings and the run could be aborted. The run time limits of Table 4.10, which correspond to median values per edge density, appear, therefore, reasonable because they cut out exactly those useless long run times. On the other hand, run times of around 150, 700, and 1500 seconds for graphs of density, respectively, 0.1, 0.5, 0.9 are enough for $\mathsf{TS}_{\mathcal{N}_1}$ to accomplish I_{max} iterations on the graphs with only few violations, that is, for those situations where it is reasonable to use all the iterations available. Note that the time limit adopted for graphs of density 0.9 appears underestimated and could be the reason for some of those feasible colouring missed in the interval $[I_{max}, 10 \times I_{max}]$ of Figure 4.24, left.

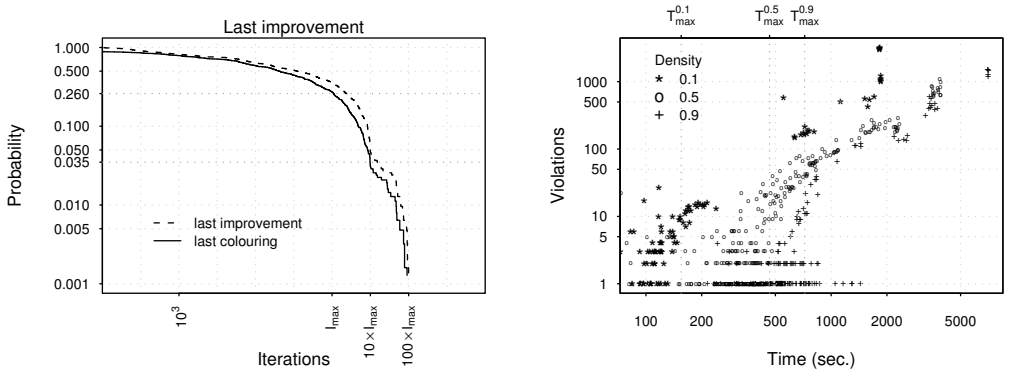


Figure 4.24.: An analysis of $\mathsf{TS}_{\mathcal{N}_1}$ in relation to the termination criterion on graphs of size 1000. $\mathsf{TS}_{\mathcal{N}_1}$ was run for $100 \times I_{max}$ iterations on each graph. On the left, we show the probability of attaining a last feasible colouring or a last improvement in solution quality in dependence with the elapsed iteration number. On the right, we show the correlation between the number of violations in the final solution and the time elapsed from a last improvement in terms of constraint violations to the end of the run, *i.e.*, the reaching of $100 \times I_{max}$ iterations. Both plots are in log-log scale.

Exact Algorithms

We run Ex-DSATUR on each of the 1260 instances using the same time limit allowed for the SLS algorithms and reported in Table 4.10. In Figure 4.25, we summarise graphically the behaviour of Ex-DSATUR. The plots give account of the time at which the best feasible colouring is found. If the colouring is proved optimal the corresponding point is plotted in black.

The first observation is that an exact solution is found either very shortly (in less than 10 seconds by our machine), or it is hardly found afterwards. Despite the large size, some graphs are easily solvable by Ex-DSATUR. Chances to find solvable graphs are highest for Geometric graphs and in particular for edge density equal to 0.5. Graphs of type G and W are solvable exactly only for very high edge density. In addition, graphs become harder to be solved exactly if the number of hidden colours increases.

We finally compared again Ex-DSATUR with BB-GCP. The result that Ex-DSATUR is better is confirmed, as it solves 147 graphs against 116 of BB-GCP. Only two graphs were solved by BB-GCP and not by Ex-DSATUR.

Construction heuristics

We compared the performance of construction heuristics considering their ranking on each single instance. For testing the statistical significance of differences, we used again all-pairwise comparisons based on the Friedman test. RLF is clearly, and by a large margin, the best algorithm on Uniform and Weight Biased graphs. For these cases, we only show the effect of edge density on the computation time of RLF for graphs of size 1000 (Figure 4.26, top). On Geometric graphs, instead, differences in solution quality with respect to DSATUR are not always statistically significant and sometimes DSATUR is better. In Figure 4.26, bottom, we report the all-pairwise comparisons with an indication of the computation time on the y-axis. No effect on the solution quality of the size of graphs was found; therefore, in the plots only graphs of size 1000 are given, on which

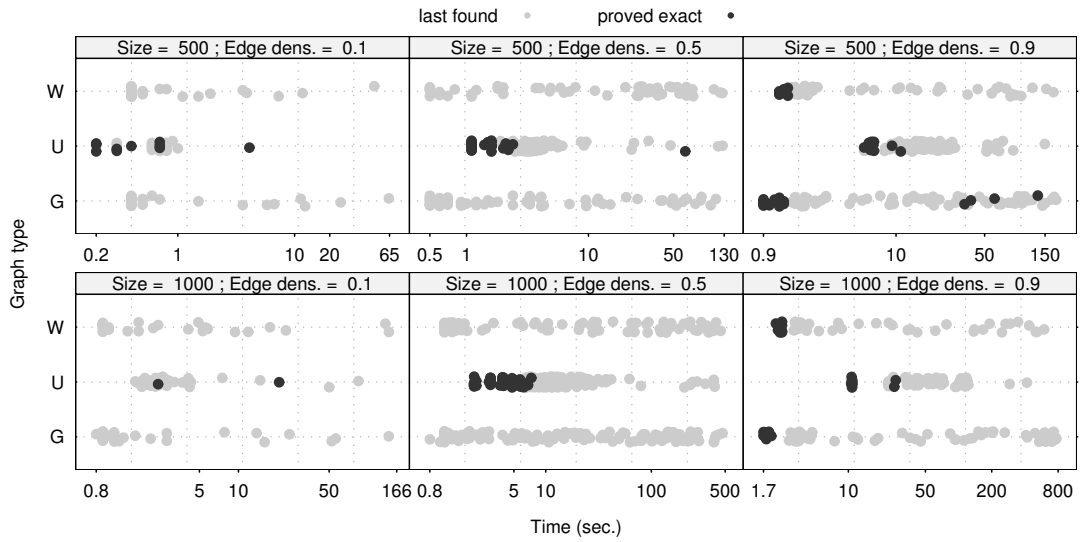


Figure 4.25.: Visual representation of the behaviour of Ex-DSATUR on the 1260 random graphs using the time limits reported in Table 4.10. For each graph, we plot a point indicating the computation times at which a last feasible colouring was found. If the solution is proved optimal the point is black, otherwise the point is grey. A jitter effect is added to the y coordinates of the data in order to make points distinguishable. The plots are logarithmic in the x-axis.

differences in computation time are more pronounced. Interestingly, there are classes of graphs in which DSATUR performs better than RLF, but unfortunately a clear pattern does not arise. A possible explanation is that RLF encounters difficulties with well balanced colourings. This conclusion is suggested by the hypothesis that these graphs have a clique number very close to the chromatic number, in which case, the variability of the size of the independent sets is rather small and not significantly influenced by the parameter "variability". In addition we observe that the performance of DSATUR with respect to RLF improves as edge density increases.

SLS algorithms

The same performance measurement adopted in Section 4.11 is also used for the analysis of this experiment.

The influence of hidden colours. As a first step in the analysis, we investigate the interaction plots between algorithm performance and hidden colours within each of the 38 classes of Table 4.8 where a colour is embedded. For several cases, the construction of a graph with the pre-specified number of colours was not possible and the generator was able to recognise the presence of a smaller upper bound (in the discussion that follows we denote with hidden colour this true upper bound, not the attempted value passed to the generator and reported in Table 4.8).

From the plots (that we do not report here) we summarise the following observations:

- In all classes, there is an interaction between algorithm performance and the number of hidden colours. However, this interaction is not strong and with few exceptions only one algorithm performs the best within the whole class. Given that we

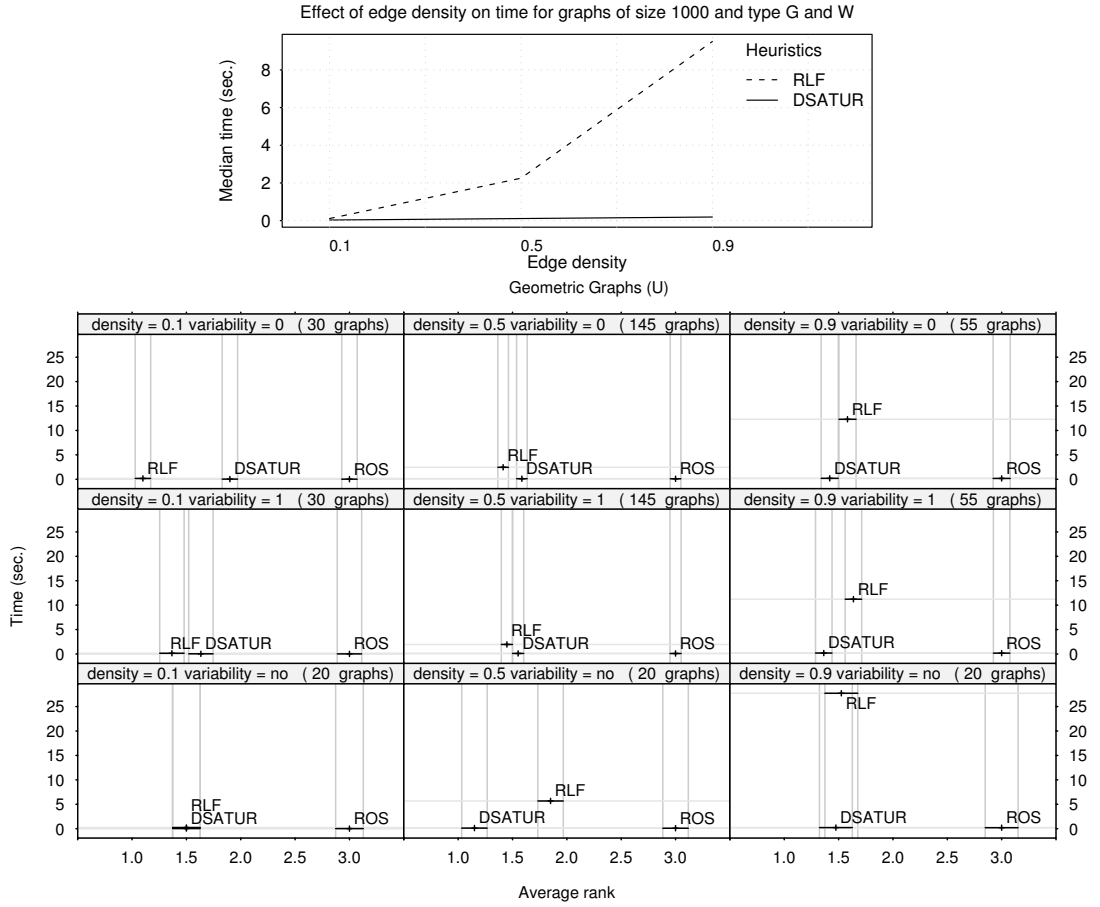


Figure 4.26.: On the upper plot the influence of edge density on the computation time of RLF and DSATUR for Uniform and Weight Biased graphs. On the lower plot, all-pairwise comparisons of construction heuristics on Geometric Graphs.

assume no *a priori* knowledge on the graphs, we neglect the interaction algorithm–hidden colours in the following analysis.

- With few exceptions, the best algorithms always have $e(\widehat{\chi}, G) = 0$ which entails that the hidden colourings are always found. Furthermore, the higher the number of hidden colours is the less precise is this bound. As an example, for the graphs with a hidden colouring of 600, the algorithms were able to find colourings with even 200 fewer colours. Therefore, the number of hidden colours cannot be used as an indicator of the hardness of solving an instance.

- On two of the 38 classes, we encounter a strange behaviour that we show in Figure 4.27. They correspond to size 1000, density 0.5, variability 0, and type of graphs G and W.

  Since hidden colouring exists, it should also be possible to reach $e(\widehat{\chi}, G) = 0$ on all those graphs. Nevertheless, the curves peak for some values of the hidden colourings, indicating that there are some values for k for which the instances are much harder to solve than usual. The region which exhibits this phenomenon arises at about 80 hidden colours for graphs of type G and between 70 and 80 for graphs of type W. It is interesting to note that this phenomenon affects also algorithms such

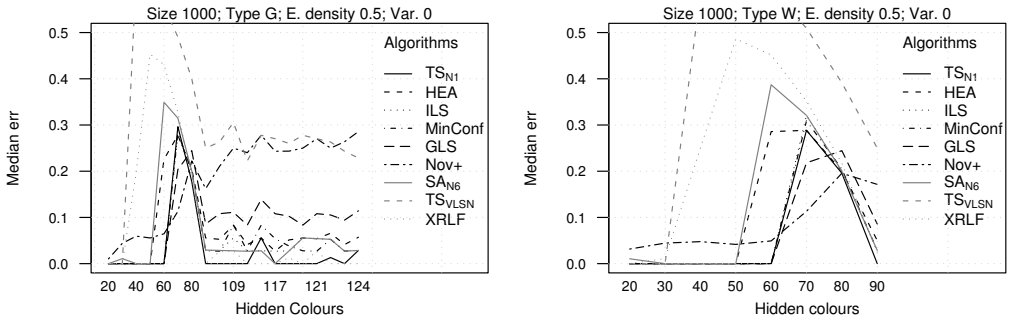


Figure 4.27.: Interaction plot between algorithm performance (expressed by the performance measure defined in Equation 4.2 with $\hat{\chi}^{best}(G)$ lower or equal to the hidden number) and the number of hidden colours. A hidden colour indicates that a feasible colouring with that on a lower number of colours exists. It is expected that all algorithms find at least one such a colouring. Nevertheless, in the region where the curves peak, no algorithm is able to find the hidden number. This indicates that in that region instances are harder to solve for all algorithms in analysis.

as $\mathsf{SA}_{\mathcal{N}_6}$ and XRLF that do not solve sequences of k-colouring problems. Moreover, the same effect was not observed in the corresponding graph classes of size 500.

A related phenomenon is the *phase transition* (or "ridge of resistance") already observed in graph colouring (Cheeseman et al., 1991; Culberson, 1992; Hogg, 1996; Culberson and Gent, 1999, 2001; Barbosa and Ferreira, 2004). In decision problems the phase transition is the passage from problems that are under-constrained to problems that are over-constrained and hence unsatisfiable. In some problems a simple parameter describing the problem structure is enough for predicting the difficulty of solving the problem and the phase transition happens in a small region of values for this parameter. In the region of the phase transition problems are typically harder to solve. This phenomenon has been shown also on the graph colouring, although, given the large number of characteristics that determine the structure of a graph, it has been impossible to link the phase transition to one specific parameter. The measure to judge the hardness of a problem on a decision problem is the computation time necessary to solve it.

Interaction between algorithms and strata on the instances. The second step of our analysis consists in identifying which stratification variables defined on the instances have an influence on algorithm performance. We report the interaction plots in Figure 4.28. As remarked in Chapter 2, interaction plots do not give a definitive answer because they do not distinguish random variations from true interaction effects. However, they might help in the understanding of the results. From Figure 4.28, it is hard to draw relevant conclusions, since the lines of median performance tend to overlap for some algorithms. However, the lines are not parallel and a strong effect, at least for some type of graphs and edge density, is observable.

The parametric model. In order to understand the significance of the factors and strata involved in the analysis, we look for a linear model with best fit and check if it is possible to apply a parametric analysis.

We tested two models: an unbalanced model, in which all data of the experiment are used, and a balanced model, in which only 10 instances from each of the 54 classes of

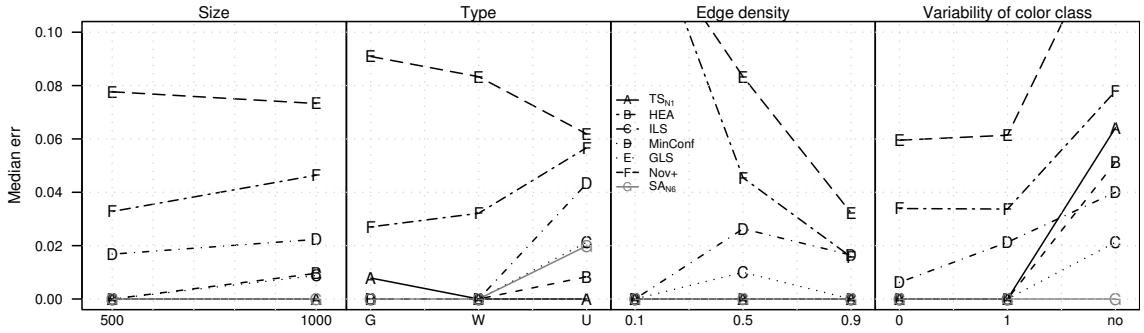


Figure 4.28.: Interaction plots between algorithms and stratification variables. The worst algorithms, XRLF and $\mathsf{TS}_{\mathsf{VLSN}}$, are removed in order to gain a clearer insight on the high performing algorithms.

	unbalanced model	balanced model
R^2	0.6111	0.6795
Interaction terms removed	$\rho(G)\times$Variability Algorithm$\times n\times$Variability Algorithm $\times$Type$\times n\times\rho(G)$ Algorithm $\times$Type$\times n\times$Variability Type$\times n\times\rho(G)\times$Variability	Algorithm$\times n\times$Variability Type$\times n\times$Variability Type$\times n\times\rho(G)\times$Variability Algorithm$\times$Type$\times n$

Table 4.11.: A synthesis of the output produced by `stepAIC`. For the balanced model an average case is reported.

Table 4.8 are bootstrapped and the procedure is repeated several times. In both cases, we considered a linear model which includes the algorithms and the 4 stratification variables on the graphs: size, type, edge density, colour class variability. Initially we included also all possible interactions among the 5 factors.

The procedure `stepAIC` in R performs an automated model selection by considering the likelihood of the linear model (Venables and Ripley, 2002). The procedure attempts to remove terms from the model by considering their significance according to the F Test. In both models, no interaction term was removed by `stepAIC` and only few resulted not significant according to the F test at the level of significance of 95%. We report a diagnostic synthesis of the models in Table 4.11 and Figure 4.29. The fact that interactions terms and main effects of the stratification variables remain in the model suggests to maintain the analysis among the instance classes separated. It must be observed that the significance of the main effects of the stratification variables entails differences in the entity of the improvements over the ROS construction heuristic (see the definition of the performance measure adopted), but this information is not enough to speculate on the hardness of solving the particular instance class. As arising from Figure 4.29, the assumptions for a parametric analysis are violated and therefore we proceed with non-parametric methods.

Non-parametric analysis. Due to the likely presence of interaction terms we present in Figure 4.30 the all-pairwise comparisons of algorithms maintaining instance classes distinct. For simplifying the presentation of the results, we assume that the size of the graphs has no influence on the relative order of the algorithms (a verified assumption, indeed). In each scenario, therefore, graphs of the two different sizes are aggregated.

Figure 4.30 presents the rank-based analysis. Permutation tests were also performed and the confidence intervals were slightly larger. We prefer therefore to base our com-

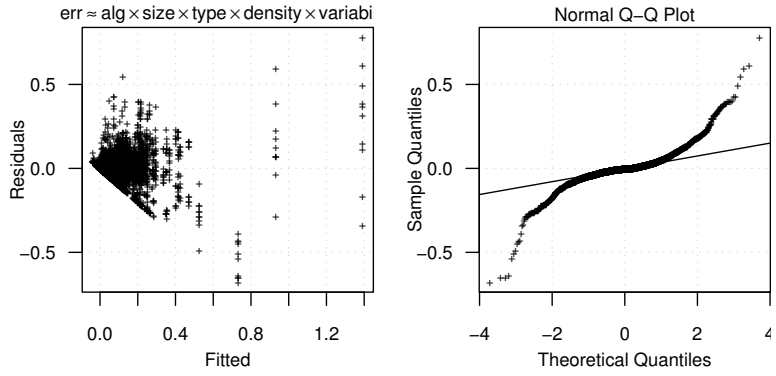


Figure 4.29.: Diagnostic plots for parametric analysis in an average case of the balanced model.

ments on the rank-based analysis, which in this case exhibits more power. For further justification on this choice we refer the reader to the Appendix B. The following are the main conclusions from the analysis.

- On Uniform graphs, $\mathsf{TS}_{\mathcal{N}_1}$ is the best algorithm on 4 scenarios and no other algorithm does significantly better. Therefore, we indicate $\mathsf{TS}_{\mathcal{N}_1}$ as the preferable method for this class. It appears particularly powerful with density 0.5. The second best is ILS which outperforms $\mathsf{TS}_{\mathcal{N}_1}$ in the scenarios with density 0.9 and variability 1.
- On the Weight Biased graphs, results are very similar to Uniform graphs. The best algorithm is $\mathsf{TS}_{\mathcal{N}_1}$, which is significantly the best in 3 scenarios, while ILS is the second best and again outperforms $\mathsf{TS}_{\mathcal{N}_1}$ on graphs with density 0.9 and variability 1.
- On the Geometric graphs, the overall best algorithm is clearly GLS. Differences from the second best are significant in 6 out of 9 scenarios, while in the other 3 scenarios it is not significantly dominated. The second best algorithm is HEA. The variability of the hidden colour classes and the edge density have a weak impact on these graphs, at least for the performance of the best algorithms, and results could be aggregated in a unique graph, in which case GLS is significantly the best algorithm.
- Scenarios with variability 0 and 1 exhibit very similar performance of algorithms suggesting that this effect is not relevant (yet, it would be interesting to verify the actual influence of this parameter on the final colour classes). An indication that the size of the colour class varies more with variability 1 is the better performance of $\mathsf{SA}_{\mathcal{N}_6}$ on these instances (recall that it performs slightly better with variability 1 than with variability 0). Furthermore, in the case of no control on the colour classes embedded in the graph, *i.e.*, no hidden colouring embedded, the results of the algorithms tend to be different for variability from 0 to 1, especially on the Uniform and Weighted Biased graphs.

The improvement over RLF. The use of SLS algorithms gives a significant improvement over the initial solution of RLF. In Figure 4.31, we show the distribution of differences between the best solution found in a graph after the application of one of the SLS algorithms and the initial RLF solution. We distinguish two main patterns which have an impact on the entity of the improvement:

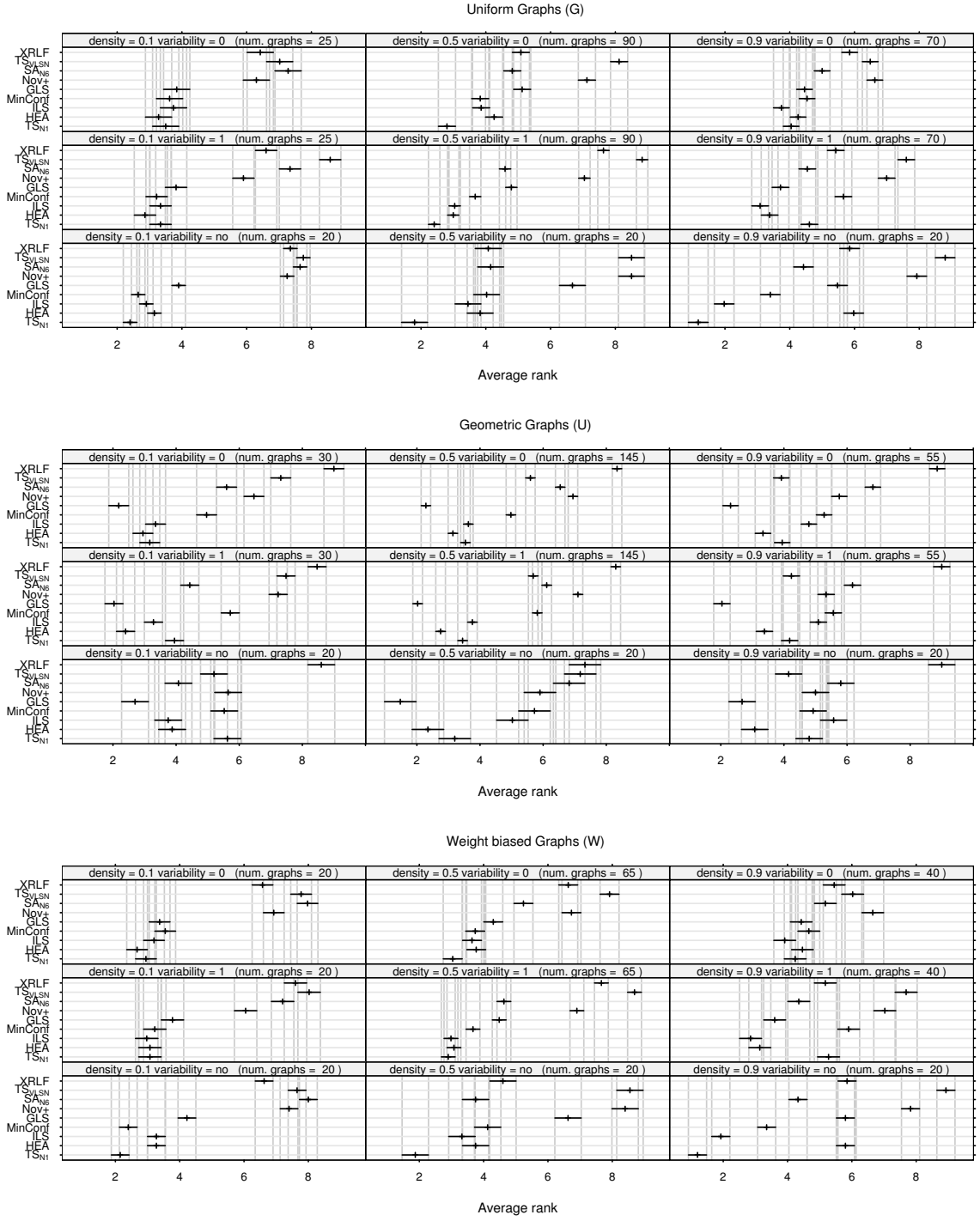


Figure 4.30.: Rank-based confidence intervals for the average ranks. The analysis is divided in three main classes of graphs: Uniform graphs, Geometric graphs, and Weight Biased graphs. Inside each class, sub-classes are determined by the combinations of the stratification variables: edge density and independent set variability. Graphs of size 500 and 1000 are aggregated and the number of instances considered is reported in the strip text of the plots.

- the improvement increases considerably with size and edge density for Uniform and Weight Biased graphs and it can reach 105 colours of improvement in the case

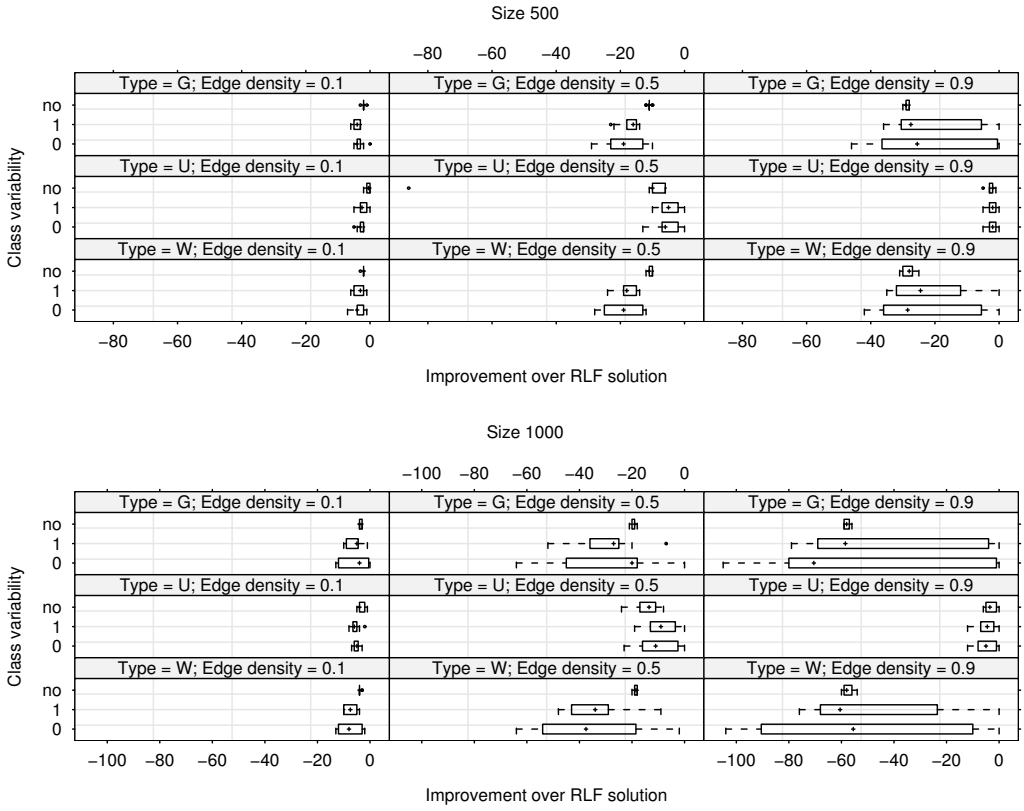


Figure 4.31.: Box-plots of differences for each graph class between the best solutions found by the SLS algorithms and the solution produced by RLF.

of Weight Biased graphs;

- in Geometric graphs, the improvement is smaller than for the other two types of graphs and more pronounced in graphs of density 0.5.

The fact that in the Geometric graphs there are not large improvements together with the results of Ex-DSATUR on those graphs let us conclude that these graphs are actually easier to solve already by RLF. For the other graphs, instead, since we do not know their optimal solution, we cannot conclude that they are harder than the Geometric graphs for SLS algorithms but we gave evidence that RLF performs poorly.

The improvement over Ex-DSATUR. Ex-DSATUR is an exact algorithm with exponential worst case. Nevertheless, it can be stopped at any time and it returns a feasible solution. The SLS algorithms here discussed have, very likely, also exponential running time, therefore, their choice is appropriate only if the approximate solution they return at a chosen time is better than the solution returned by Ex-DSATUR in the same run time. In Figure 4.32, we give evidence that the SLS algorithms are impressively better than Ex-DSATUR. Improvements of even more than 150 colours over Ex-DSATUR can be attained by using these methods. Interestingly, Ex-DSATUR does even worse than RLF for several instances. In the case of edge density 0.9, graphs of type W, and size 1000, RLF finds in some cases even 100 colours less, while very few are the cases where Ex-DSATUR is better. The reason why Ex-DSATUR is worse than RLF is related to the comparison between the two heuristics RLF and DSATUR.

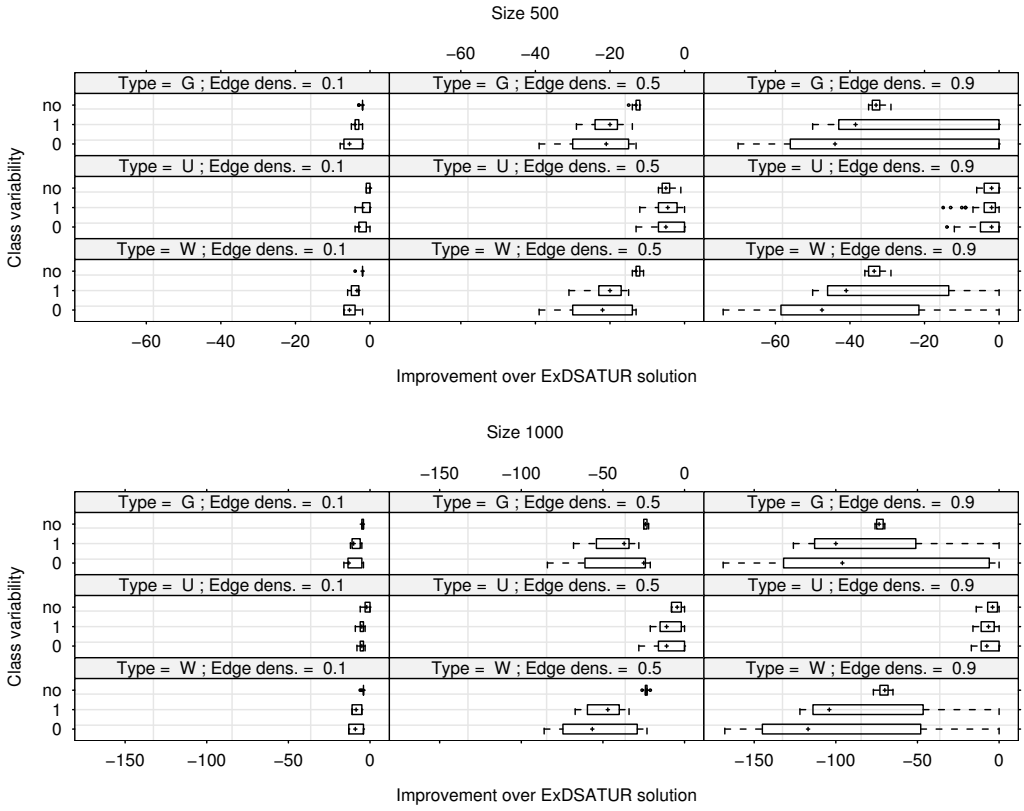


Figure 4.32.: Box-plots of differences for each graph class between the best solutions found by the SLS algorithms and the solution produced by the Ex-DSATUR heuristic. If only $\mathsf{TS}_{\mathcal{N}_1}$ among the SLS algorithms is considered the graph remains substantially the same.

4.14. Discussion

We compared SLS algorithms for large graph colouring instances. We showed that exact algorithms can solve graphs until about 100 vertices, in the general case, while for *Geometric random graphs* it may be possible to solve exactly even instances with 1000 vertices.

We discussed graph reduction rules which can be used to remove vertices that are easily coloured independently by the colours assigned to the other vertices. In addition, we gave a computational analysis of three simple and fast construction heuristics. RLF appears to be the best heuristic, although its running time is higher than that for the other two heuristics. If a very fast heuristic is needed, then DSATUR is the best choice. DSATUR is furthermore not always dominated by RLF on *Geometric random graphs*. In some cases, the reduction rules together with construction heuristics are enough to find the chromatic number. Graphs that are exactly colourable in this way from the DIMACS repository are those from register allocation, algebraic Quasigroups of size smaller than 100, graphs from the Donald Knuth's Stanford GraphBase, and Mycielski graphs. When reduction rules and construction heuristics are not enough, the solution quality can be considerably improved in relatively short computation time with the application of SLS methods.

Our important contribution for the advancement of SLS methods for graph colouring has been an in-depth analysis of different neighbourhood structures for Iterative Im-

provement. In particular, we developed and analysed a *very large scale neighbourhood* (VLSN) which was for the first time applied to graph colouring. This neighbourhood is based on cyclic and path exchanges which can involve more than one vertex from different colour classes. An effective examination of this neighbourhood is possible thanks to the favourable correspondence between the problem of finding profitable cyclic exchanges and the problem of finding subset disjoint negative cost cycles in an improvement graph (Thompson and Orlin, 1989). This model has also been applied to other problems: the capacited minimum spanning tree problem (Ahuja et al., 2001a), the single source capacited facility location problem Ahuja et al. (2004), and the multi-resource generalised assignment problem. In all these cases VLSN was enhanced with basic SLS methods and results were encouraging, producing best known results in a few cases. In contrast, the application of VLSN to the vehicle routing problem with time windows studied by Ibaraki et al. (2004) was not profitable. In our case we adapted an algorithm for the examination of the neighbourhood which is due to Dumitrescu (2002). We proved that our new neighbourhood entails fewer local optima in the GCP and therefore Iterative Improvement methods with cyclic and path exchanges can achieve higher performance then simple one-exchange neighbourhoods. Nevertheless, when high performances on large size graphs are required, the computational cost of examining the larger neighbourhood can have a negative impact. Therefore, we introduced heuristic truncation rules which worsen only slightly the quality performance, while decreasing considerably the complexity of the examination. Finally, we enhanced Iterative Improvement with the Tabu Search metaheuristic and tested the algorithm empirically on large graphs.

From the experimental analysis it arose clearly that the use of VLSN is ineffective, despite its good analytic properties. The time spent for examining the neighbourhood is better used by Tabu Search for exploring new regions of the search space. Apparently, Tabu Search alone is already a good mechanism in GCP for coping with the small one-exchange neighbourhood limited to conflicting vertices. We mention that, almost contemporaneously to our research, another attempt to adapt to graph colouring exponentially large neighbourhoods that can be searched polynomially was developed by Glass and Prügel-Bennett (2005). Although their neighbourhood structure is totally different from our cyclic and path exchange neighbourhoods the final outcome is the same: the one-exchange neighbourhood remains preferable. Hence, even highly promising neighbourhoods can perform poorly in practice and only computational experimentation can ascertain their competitiveness. Reporting about such negative results is therefore important.

The experimental analysis was then extended to other SLS methods. Some of them are the best known methods for the GCP, others have been applied here for the first time. The newly applied methods are Iterated Local Search, Novelty$^+$, and Guided Local Search. The choice for these methods is due to their very positive results in problems similar to the GCP.

Two experiments were designed to evaluate SLS algorithms. A first experiment was conducted on the instances of the DIMACS repository. In this case, 80 graphs remain after the removal of the easy ones. These graphs are unevenly distributed among different classes, and this fact compelled us to maintain separated the analysis among the classes. A second experiment was designed on randomly generated graphs. In this case, we could increase the number of instances thus allowing to better reduce the variance of the performance estimates. We considered a total of 1260 graphs that were constructed with different structural parameters in order to gain a better insight into the relationship

between algorithms and graph structure. We took into account size, edge density, type of graph, and characteristic of embedded colour classes. These are all features that may also occur in a similar form in practical contexts.

The results of the two experiments present some differences. On the *Uniform and Weight Biased random graphs*, results of the second experiment are much more conclusive than results on the DIMACS instances. We deem this an indication that the experimental design adopted in the second experiment is more appropriate. In this case, $\mathsf{TS}_{\mathcal{N}_1}$ is the best algorithm with graphs of density 0.5 and remains not dominated at edge density of 0.1 and 0.9.

On *Geometric random graphs*, GLS is clearly the best algorithm. This result is similar to the result obtained for the WAP graphs. These two classes of graphs have the similar characteristic of having the clique number and the chromatic number very close and the result leads us to conjecture that GLS works well for graphs with this property. For the WAP graphs, we know that there is a clique of at least 40 vertices in each instance as they arise from the configuration of an optical network (in all instances, there exist fibres that contain 40 light-paths, and light-paths sharing the same fibre are mutually adjacent, A. Koster, priv. comm.). For four of these graphs the optimal solution is known but GLS reaches it only in one case. This provides further indication that these graphs might be challenging also for future comparisons. *Weight Biased graphs* are, instead, graphs in which large cliques are avoided and indeed GLS is performing poorly.

These results are interesting and new. They show that $\mathsf{TS}_{\mathcal{N}_1}$ is not clearly dominated by other approaches which were claimed better (like HEA) if the computation times are not too large and the parameter settings are not fine-tuned on each instance. XRLF, also believed to be quite effective, was shown to perform poorly on all graphs except on the Queens graphs. Moreover, this algorithm is very unpredictable in its behaviour and, despite the indications presented for its tuning, we do not recommend its use. We found three possible explanations for the difference of these results from previously published comparisons: (i) algorithms that were fine-tuned to the specific k of the k-colouring problem to solve; (ii) comments on results were based on best results found, which is known from statistics to be an incorrect way for making some inference; (iii) comments were based on too few instances (in the case of XRLF mainly on one single instance); (iv) the run times considered were much longer (although this claim should be restricted to HEA only, which, as we verified, brings its contribution only later in the search on Uniform random graphs); (v) and *a priori* knowledge was used for determining a good starting value of k, close to the best known solutions (we gave, instead, confirmation that for few graphs, like the Flat graphs, it may be harder to solve the problem at a value of k quite distant from $\chi(G)$ than for values close to it).

Finally, we showed that the use of SLS algorithms is worthwhile as it can largely improve the solutions found by the best construction heuristics and one of the best exact algorithms if this is stopped prematurely at the same time as the SLS algorithms. In both cases the improvements are impressive with peaks of about 100-150 colours gained for high density graphs of size 1000.

In conclusion, we shed light on which are the best algorithms for solving the GCP. The indication arose is that Tabu Search and Guided Local Search using the RLF construction heuristic and a local search based on the one-exchange neighbourhood are state-of-the-art approaches for large size graphs, as they are not dominated by any other SLS algorithm and often attain the best results. These are quite general purpose algorithms which do not exploit problem specific knowledge. Further research should be carried out on ad-

hoc algorithms for specific types of graphs, where problem specific knowledge could be used to boost the performances.

Chapter 5.

Graph Colouring Generalisations

In which we introduce some generalisations of graph colouring with relevant real life applications. We focus on Stochastic Local Search methods for solving the set T-colouring problem used for modelling the assignment of frequencies in mobile networks.

5.1. Introduction

In the previous chapter, we studied the classical graph colouring problem (GCP) on general graphs. Yet, the GCP may not be sufficient for modelling real life situations, as there may be further constraints to consider besides those representable by simple edges. An example is the assignment of frequencies in the design of mobile phone networks. Radio stations are placed in the plane and their transmissions cover given regions, called "cells". Each station transmits at a certain frequency but due to interferences, stations whose cells are geographically close to each other, cannot use the same frequencies. This situation might look at first sight as the problem of colouring a planar graph, but in practice things are more complicated. The available frequency band is partitioned into a set of channels with equal bandwidth. The available channels, which may be seen as colours, are denoted by integer numbers. Typically, not only prohibitions to use the same channel (co-channel constraints) apply, but also adjacent channels must be forbidden if they cause excessive noise. Interference constraints between transmitters impose, therefore, frequencies to be different and to maintain certain separations on their values from F. Moreover, these constraints may involve also cells which are not contiguous, thus making the graph not anymore planar. Finally, transmitters can require more than one frequency if the number of mobile phones to serve in a cell is high.[1] These new constraints on the separation of frequencies and multiple frequency assignments to a vertex are modelled by generalisations of graph colouring which are called *T-colouring* and *set T-colouring*. As we will see, the definition of these problems is not unique since constraints can assume different special connotations and several criteria to evaluate a colouring can be taken into account in the optimisation version.

Another example where a generalisation of graph colouring is needed, is course timetabling, in which we have to schedule lectures in time periods by taking into account that some pairs of lectures cannot be scheduled at the same time because they share students, lecturers or rooms. Again, this may look like a GCP but in practice, timetabling, is never

[1] This problem is often also referred to as Fixed Channel Assignment to emphasise that the model is static, *i.e.*, the set of connections remains stable over the time, or Radio Link Frequency Assignment, to denote the specific field of application.

formulated so straightforwardly and other constraints are commonly present. Realistic cases include, for example, the presence of restrictions on the assignment of lectures to certain, specific time periods or the preassignment of some lectures due to particular needs of the lecturers. The first case can be formalised as a *list colouring* problem while the second as a *precolouring extension* problem.

In this chapter, we study the application of SLS methods for solving such generalisations of graph colouring. We start by re-implementing known approaches and by comparing them rigorously. Besides this, we extend algorithms that have been shown successful in the GCP, develop other original ideas, and indicate the best approach to deal with the new types of constraints in graph colouring generalisations. The systematic study on these problems provide important insights that turned out to be useful in Chapter 6 when studying a complex timetabling problem.

We first pose the definition of some of the graph colouring generalisations which are relevant for our study and mention some basic theoretical results in Section 5.2. We then place our work into the context of recent literature and introduce the benchmark instances which motivated our research in Sections 5.3 and 5.4. In Section 5.5, we present results on the precolouring extension while in Section 5.6 the main attention is focused on an extensive experimental comparison of algorithms for the set T-colouring problem. A final discussion, which summarises the main findings of our study, concludes the chapter.

5.2. Formal definitions

In this section, we define formally some of the graph colouring generalisations. We proceed in increasing order of constraint complexity arriving at the definition of the *set T-colouring* problem, which is the central problem of this chapter.

5.2.1. Precolouring Extension

In the precolouring extension, a vertex subset $W \subseteq V$ of a graph G is *precoloured*, that is, its vertices are assigned unchangeable colours. The task is then to extend the colouring to the *uncoloured* vertices. Formally, the problem is defined as follows:

> ***Input:*** *A undirected graph $G = (V, E)$, a set of colours Γ with $|\Gamma| = k \leq V$, a subset of vertices $W \subseteq V$, and a colouring of W, $\varphi_W : W \to \Gamma$.*
>
> ***Question:*** *Is there a feasible k-colouring $\varphi : V \setminus W \to \Gamma$ such that $\varphi(u) \neq \varphi(v)$ whenever $u, v \in V \setminus W, (u, v) \in E$ and $\varphi(u) \neq \varphi_W(v)$ whenever $u \in V \setminus W, v \in W, (u, v) \in E$?*

The k-colouring φ is called *extension* of the precolouring φ_W.

5.2.2. List Colouring

A list colouring problem arises when each vertex has associated specifications on the colours that are *admissible*. Formally, it can be expressed as:

Input: *An undirected graph $G = (V, E)$ with vertex set $V = \{v_1, v_2, \ldots, v_n\}$, a set of colours Γ, and a list colour assignment $\mathcal{L} = (L_1, \ldots, L_n)$ with $L_i \subseteq \Gamma,\ \forall 1 \leq i \leq n$.*

Question: *Is there a vertex colouring φ such that:*

- *$\varphi(v_i) \in L_i$ for all $1 \leq i \leq n$, and*
- *$\varphi(v_i) \neq \varphi(v_j)$ for all $(v_i, v_j) \in E$?*

The mapping $\varphi : V \rightarrow \Gamma$ is also called *list colouring*, or *$\mathcal{L}$-colouring*. It is then possible to distinguish between *feasible list colourings*, *i.e.*, list colourings that answer the problem positively, and *infeasible list colourings*. Typically, $\Gamma \subset \mathbb{N}$ and $\Gamma = L_1 \cup \ldots \cup L_n$. Each L_i defines the set of *admissible* colours for vertex v_i (often, the set of *forbidden* colouring is given in its place).

If $|L_i| = k$ for all i, then $\mathcal{L}$ is termed a *k-assignment*. The *choice number* of G (also called in the literature *list chromatic number*), denoted by $\chi_l(G)$, is the smallest k such that every k-assignment $\mathcal{L}$ admits a feasible list colouring. G is then said to be *k-choosable* if $\chi_l(G) \leq k$. Similarly to the GCP, it is possible to define an optimisation version of the list colouring problems which is known as the *k-choosability* problem.

Input: *An undirected graph $G = (V, E)$ and a set of colours Γ.*

Question: *Which is the smallest k such that every k-assignment $\mathcal{L}$ with $L_i \subseteq \Gamma$ and $|L_i| = k \in \mathbb{N}$ for all i admits a feasible list colouring?*

Note that the chromatic number problem can be seen as a k-choosability problem in which the lists in the k-assignment are all identical $L_i = \{1, \ldots, k\}$. It follows that $\chi(G) \leq \chi_l(G)$.

5.2.3. T-Colouring

In the T-colouring problem, a set of disallowed separations between colours assigned to adjacent vertices is associated to each edge in E. Without loss of generality, the colour separations T_{uv} may be expressed in terms of non-negative integers, that is, forbidden differences between elements of the set of colours $\Gamma \subset \mathbb{N}$. Formally, the problem is defined by:

Input: *An undirected graph $G = (V, E)$, a set of colours Γ with $\Gamma \subset \mathbb{N}$, and a collection of sets $\mathcal{T} : \{T_{uv} \subset \mathbb{N}_0,\ \forall (u, v) \in E\}$.*

Question: *Is there a k-colouring $\varphi : V \rightarrow \Gamma$, such that $|\varphi(u) - \varphi(v)| \notin T_{uv}$, $\forall u, v \in V$?*

A k-colouring that fail to satisfy some of the separation constraints is an infeasible *T-colouring*; otherwise it is feasible. Note that if $T_{uv} = \{0\}$ for all $(u, v) \in E$, then a T-colouring problem reduces to the GCP.

The optimisation version of this problem focuses on four characteristics of a T-colouring: the number of *edge conflicts*, in case of infeasible T-colouring; the *order* of the T-colouring, defined as the number of different colours effectively used in the T-colouring; the *span* of the T-colouring, that is, the difference between the largest and the smallest colour used, *i.e.*, $\max_{u,v \in V}(|\varphi(u) - \varphi(v)|)$; and the *edge span* of the T-colouring, that is, the largest difference between colours at any edge, *i.e.*, $\max_{(uv) \in E}(|\varphi(u) - \varphi(v)|)$. Distinguishing the four cases, we obtain four optimisation problems:

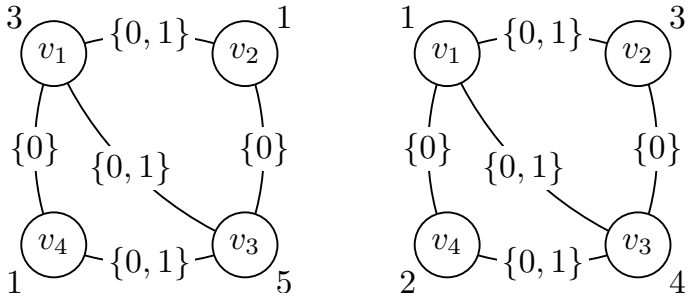


Figure 5.1.: T-colouring problem with $\chi_T(G) = 3$ and $sp_T = (G) = 3$.

Input: *An undirected graph* $G = (V, E)$*, a set of colours* Γ *with* $\Gamma \subset \mathbb{N}$*, and a collection of sets* $\mathcal{T} : \{T_{uv} \subset \mathbb{N}_0,\ \forall (u, v) \in E\}$.

Question: *One of the following:*

Minimal number of conflicts: Which is the T*-colouring that minimises the number of conflicting edges?*

T-chromatic number: Which is the smallest order $\chi_T(G)$*, known as* T*-chromatic number (or* T*-order), such that a feasible* T*-colouring exists?*

T-span: Which is the smallest span $sp_T(G)$*, known as* T*-span, such that a feasible* T*-colouring exists?*

Edge T-span: Which is the smallest edge span $sp^e_T(G)$*, known as edge* T*-span, such that a feasible* T*-colouring exists?*

Note that with $\Gamma = \{1, \ldots, k\}$, the span of the graph corresponds to $k - 1$ and therefore minimising the span corresponds to minimising k. Moreover, in the case $T = \{0\}$ we have $\chi_T(G) = \chi(G)$, $sp_T(G) = \chi(G) - 1$, and the concept of edge span reduces to the concept of *graph bandwidth*.

Not much work has been done for understanding possible relations between the three last criteria. Hale (1980) points out that there are cases where no optimal order T-colouring gives an optimal span and vice-versa. For instance, if the graph G and the sets $\mathcal{T}$ are those of Figure 5.1, then $\chi_T(G) = 3$ and $sp_T(G) = 3$ but a T-colouring with order 3 has span at least 4, whereas a T-colouring with span 3 has order 4.

This example should raise the interest for the study of the trade off between the two objectives of minimising the order and the maximising span of the T-colouring. Hale (1980) defines the *restricted colouring problem*[2] as the attempt to minimise the span (order) while maintaining fixed the order (span). A special case of restricted colourings occurs when attempting to minimise the span (order) for the order fixed to $\chi_T(G)$ (span fixed to $sp_T(G)$). But no other relevant work appeared in these directions.

The case in which all $T_{uv} \in \mathcal{T}$ consist of consecutive integers $\{0, 1, \ldots, t_{uv} - 1\}$ for all $(u, v) \in E$ is more relevant for practical applications. This particular version of the problem is known as *separation distance T-colouring* problem and the values $t_{u,v}$ associated with each edge $\{u, v\}$ are called *colour distances* (Eisenblätter et al., 2002). The constraints related with $\mathcal{T}$ that a T-colouring must satisfy become then $|\varphi(u) - \varphi(v)| \geq t_{uv},\ \forall (u, v) \in E$.

[2]The term *restricted* is used in graph colouring with more than one meaning. Besides the meaning explained in the present text, Dorne and Hao (1998b) use the term to indicate the special case of T-colouring where each $T_{uv} \in T$ is a set of consecutive integers of the form $T_{u,v} = \{0, 1, 2, \ldots, t_{u,v} - 1\}$.

Finally, the problem of minimising edge conflicts in the T-colouring is strictly related to the *minimal interference problem*, which has practical relevance in frequency assignment (Aardal et al., 2001; Eisenblätter et al., 2002). This is the case, when penalties are associated to disallowed separations and acceptable interferences are weighted differently from forbidden interferences. This gives rise to the minimal interference formulation where the total penalty has to be minimised.

5.2.4. List T-Colouring

Combining the two previous problems, we obtain the list T-colouring problem, introduced by Tesman (1993), which may be formalised as follows:

> ***Input:*** *An undirected graph $G = (V, E)$ with vertex set $V = \{v_1, v_2, \ldots, v_n\}$, a set of colours Γ, $\Gamma \subset \mathbb{N}$, a list colour assignment $\mathcal{L} = (L_1, \ldots, L_n)$ with $L_i \subseteq \Gamma$, $\forall 1 \leq i \leq n$, and a collection of sets $\mathcal{T} : \{T_{uv} \subset \mathbb{N}_0,\ \forall (u, v) \in E\}$.*
>
> ***Question:*** *Is there a T-colouring in which each vertex $v_i \in V$ receives a colour $\varphi(v_i)$ from the corresponding list of admissible colours L_i?*

The T-colouring problem can be seen as a particular case of the list T-colouring problem where all $L_i \in \mathcal{L}$ are identical.

Each of the four optimisation criteria defined for the T-colouring problem can be used for list T-colourings. The formalisation is the straightforward extension of the optimisation case of T-colouring problem, hence we omit it.

An additional optimisation problem which is studied with list T-colourings, is finding the *T-choice number*:

> ***Input:*** *An undirected graph $G = (V, E)$, a set of colours $\Gamma \subset \mathbb{N}$, and a collection of sets $\mathcal{T}$.*
>
> ***Question:*** *Which is the smallest k such that every k-assignment $\mathcal{L}$ with $|L_i| = k \in \mathbb{N}_0$, admits a feasible list T-colouring?*

5.2.5. Set T-Colouring

A further extension of the T-colouring problem, also originally described by Tesman (1990), is the set T-colouring problem defined as follows:

> ***Input:*** *An undirected graph $G = (V, E)$, a set of colours $\Gamma \subset \mathbb{N}$, $|\Gamma| = k$, a number of required colours $r(v)$, $\forall v \in V$, and a collection of sets $\mathcal{T} : \{T_{uv} \subset \mathbb{N}_0$, $\forall (u, v) \in E$ and $T_{uu} \subset \mathbb{N}_0\ \forall u \in V\}$.*
>
> ***Question:*** *Is there a vertex colouring corresponding to a multi-valued function $\varphi : V \rightarrow \Gamma$ such that each vertex receives a set of colours $\varphi(v) = S(v) \subseteq \Gamma$ and:*
>
> - *$|S(v)| = r(v)$ for all $v \in V$;*
> - *$|\varphi(v, i) - \varphi(v, j)| \notin T_{vv} \quad \forall \varphi(v, i), \varphi(v, j), i \neq j$; and*
> - *$|\varphi(v, i) - \varphi(u, j)| \notin T_{uv} \quad \forall (u, v) \in E,\ \forall \varphi(v, i) \in \varphi(v),\ \varphi(u, j) \in \varphi(u)$?*

We call such multi-valued colourings *set T-colourings* and denote them as feasible if they constitute an answer to the problem, and infeasible otherwise. The three constraints that a set T-colouring must satisfy in order to be feasible are called *requirement constraints*, *vertex constraints*, and *edge constraints*, respectively.

Generalising the characteristics of T-colourings, we can define the *order* of a set T-colouring to be the number of distinct integers used in all the sets $S(v)$ and the *span* to be the difference between the largest and the smallest integers used in any of the sets $S(v)$. We denote as $\chi^s_T(G)$ the minimal order and as $sp^s_T(G)$ the minimal span of a set T-colouring of G.

All the optimisation versions of the T-colouring problem can be extended to the case of sets of colours assigned to vertices. However, three received major attention in the literature: in case a feasible set T-colouring exists, the *minimal order*, and the *minimal span* problem; in case no feasible set T-colouring exists, the *maximal service*. This latter version arises mainly in the field of frequency assignment and seeks for a partial set T-colouring which assigns to the vertices as many colours as possible while maintaining feasibility. Basically, in this case the requirement of colours at each vertex is relaxed to express a preference rather than a strict restriction. Formally, the three optimisation versions are expressed as follows.

Input: *An undirected graph $G = (V, E)$, a set of colours $\Gamma \in \mathbb{N}_0$, $|\Gamma| = k$, a number of required colours $r(v)$ for each vertex and a collection of sets $\mathcal{T}$: $\{T_{uv} \subset \mathbb{N}_0$, $\forall (u, v) \in E$ and $T_{uu} \subset \mathbb{N}_0 \; \forall u \in V\}$.*

Question: *One of the following:*

Maximal service: Which is the set T-colouring that satisfies all vertex and edge constraints and maximises the number requirement constraints satisfied?

Minimal order: Which is the smallest order $\chi^s_T(G)$ such that a feasible set T-colouring exists?

Minimal span: Which is the smallest span $sp^s_T(G)$ such that a feasible set T-colouring exists?

Combining the ideas of set T-colouring and list T-colouring, the notion of set list T-colouring can be straightforwardly derived. This version of graph colouring is, however, not relevant to the scope of this thesis.

5.2.6. Related problems

A special case of set colourings arises when each $r(v) = r > 0$ for all $v \in V$. In that case, the assignment of colours is called an *r-tuple colouring*. The *r-tuple chromatic number*, denoted by $\chi_r(G)$, is defined as the minimal number of distinct colours in a r-tuple, that is, $\chi_r(G) = \min\{|\bigcup_{v \in V} S(v)|\}$ such that a feasible r-tuple colouring exists. Order and span of an r-tuple can also be straightforwardly redefined.

Another special case of set colouring is the *interval k-colouring* in which each set $S(v)$ is made of $r(v)$ consecutive colours (de Werra, 1990). This problem is particularly relevant for applications like course timetabling and employee timetabling when lessons or shifts can be of variable length but must be scheduled in consecutive hours.

Often, in graph theory the T-colouring problem is treated with the collection $\mathcal{T}$ reduced to one single set T, which remains the same for all the edges. The general case can be obtained by considering l different graphs, $G_0, G_1, \ldots, G_l$ each on the same vertex set V with the following properties: $G_0 \subseteq G_1 \subseteq G_2 \subseteq \ldots \subseteq G_l$ and $T(0) \subseteq T(1) \subseteq T(2) \subseteq \ldots \subseteq T(l)$. The task becomes finding a colouring φ that assigns to each vertex a colour, such that φ is simultaneously a $T(i)$-colouring of G_i for all $i = 1, \ldots, l$, that is, $(u, v) \in E(G_i) \Rightarrow |\varphi(u) - \varphi(v)| \notin T(i),\ i = 0, 1, \ldots, l$.

With respect to list colouring, Mahadev and Roberts (2003) have recently introduced new formulations which can find relevant applications in the field of timetabling and physical mapping of DNA. Namely, given a graph G and $\mathcal{L}$ for which a list colouring does not exist, one can be asked for the minimal modifications to G or to $\mathcal{L}$ in order to make them list colourable. In the first case, one may be interested in knowing what is the smallest number of edges to remove for having a feasible list colouring. In the second case, one may look for the minimal number p of vertices, $v_1, v_2, \ldots, v_p$ such that a colour must be added to the respective set of admissible colours $L_i, i = 1, 2, \ldots, p$. Alternatively, one can define a *trade* from vertex v to vertex u as the operation of removing one colour c from L_v and adding it to L_u, and seek how many trades are needed to make the list assignment list colourable.

5.2.7. Known theoretical results

The decision versions of the T-colouring, list colouring and set T-colouring problems are $\mathcal{NP}$-complete for general graphs. This is due to the fact that the k-colouring problem can be expressed as a particular case of these problems. Intuitively, results for the optimisation problems are strictly related to the type of the graph G and the set $\mathcal{T}$. General results may therefore only be quite weak. One such result is the following.

Theorem 5.1 (Hale, 1980) *For all graphs G and sets $\mathcal{T}$, $\chi_T(G) \leq sp_T(G)$.*

If all T sets are the same, *i.e.*, $\mathcal{T} = \{T \subset \mathbb{N}, \forall (u, v) \in E\}$, the following theorems hold on all general graphs.

Theorem 5.2 (Cozzens and Roberts, 1982) *For all graphs G and sets T,*

$$\chi_T(G) = \chi(G).$$

Theorem 5.3 (Cozzens and Roberts, 1982) *For all graphs G and sets T,*

$$\chi(G) - 1 \leq sp_T(G) \leq |T|(\chi(G) - 1).$$

Theorem 5.4 (Cozzens and Roberts, 1982) *For all graphs G and sets T,*

$$sp_T(K_{\omega(G)}) \leq sp_T(G) \leq sp_T(K_{\chi(G)}),$$

where $\omega(G)$ is the size of the largest clique of G and K_n is a complete graph of size n.

In particular, the last theorem stimulated research on $sp_T(K_n)$ when the typology of T is known. These results are, however, not valid for the more general case of $\mathcal{T}$ as a collection of different sets.

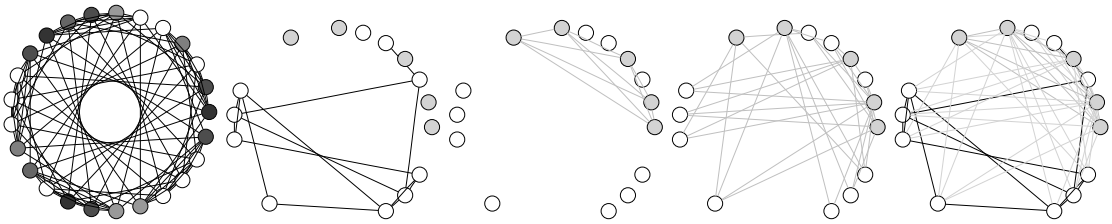


Figure 5.2.: The transformation of the precoloured graph `qwhdec.order5.holes10`. The first graph on the left is the original graph with 15 precoloured vertices and 10 "holes" in white. The precoloured vertices and the edges incident to them are removed and in their place one single vertex (represented in light grey) for each colour used is inserted in the graph (second graph). The vertices added are connected each other forming a clique (third graph), then the new vertices are connected to the vertices left in the graph if one of the vertices they represent was connected to such a vertex in the original graph (fourth graph). The final graph is composed by the edges and the vertices added (fifth graph).

For a review on more precise theoretical results on special cases of G and $\mathcal{T}$ we refer to Roberts (1991) and Murphey et al. (1999). Here, since special cases of T-colouring problems are rare in practice (Aardal et al., 2001), we orient our research to the general cases.

5.2.8. Problem transformations

Precolouring Extension into Chromatic Number

Let $G = (V, E)$ be a graph with precoloured set W and let k be a bound on the number of colours. Assuming that $W_i \subseteq W$ is the (possibly empty) set of vertices of colour i for $1 \leq i \leq k$, it is possible to replace each W_i in G by a new vertex u_i and join $v \in V \setminus W$ to u_i if and only if v had at least one neighbour in W_i. All new vertices u_i are then made mutually adjacent, forming a clique of order k. The precolouring of G is extendable with colour bound k if and only if the modified graph has chromatic number k. An example of the construction of the modified graph is shown in Figure 5.2.

Set T-Colouring into T-Colouring

A graph $G = (V, E)$ with a collection of sets $\mathcal{T}$ of constraints and $r(v), \forall v \in V$ colour requirements can be transformed into a graph $G^S = (V^S, E^S)$ with $r(v) = 1, \forall v \in V^S$ by creating a vertex u for each requirement of a vertex $v \in V$ so that at the end $|V^S| = \sum_{v \in V} r(v)$. The vertices $u \in V^S$ derived from a vertex $v \in V$ form a clique of order $r(v)$ in which each edge receives the constraint T_{vv}. Every such vertex is then connected with each vertex of the clique induced by another vertex $w \in V$ if $(v, w) \in E$. The colour separation associated with these edges is T_{wv}. In the next sections we will often refer to this transformation by naming the graph G^S the *split graph*.

5.3. State of the art and motivations

The problems introduced above have received particular attention in recent years both from a theoretical and application perspective. Graph theorists are currently particularly involved in graph colouring generalisations. Many properties of these problems are studied, such as complexity, bounding, and approximations. Moreover, the GCP generalisations introduced are just a few of those included in the family of graph colouring problems and the interest is also in classifying and discovering new graph colouring problems with practical relevance. Surveys that collect formulations of many such problems are those of Jensen and Toft (1995) and Tuza (1997), as also the thesis of Marx (2004). From the original papers of Roberts (1991) and Tesman (1990, 1993) the attention has moved to specific graph typologies and specific problem formulations and the literature has grown rapidly. For a bibliography of related publications we refer the interested reader to the web site of the DIMACS research centre which besides the "Computational Challenge" is also holding a series of workshops on theoretical results on graph colourings and their generalisations.[3]

As far as the T-colouring problem is concerned, Giaro et al. (2003a) and Giaro et al. (2003b) give more precise results on its complexity. Lower bounds have been investigated primarily for the assessment of frequency assignment algorithms (see Montemanni (2001); Allen et al. (1999); Janssen and Wentzell (2000); Smith et al. (2000)), while few approximation results are available (among them Simon, 1989 and Janssen and Narayanan, 2001). Finally, publications on approximate approaches are those of Costa (1993), Dorne and Hao (1998b), Phan and Skiena (2002), Lim et al. (2003) and Prestwich (2003); the latter two present results on the instances proposed in the context of the DIMACS "Computational Challenge". We will mainly focus on these references, describing and reimplementing the approximate methods proposed.

In contrast, algorithms for the frequency assignment problem (FAP) have been well studied in the last three decades. Several resources are collected on the web,[4] among them surveys (Eisenblätter et al., 2002; Aardal et al., 2001), an updated bibliography, and sets of benchmark instances. Exact and approximate algorithms have been proposed and tested on instances from real applications. Particularly famous is the Philadelphia instance, made of 21 vertices, which allows the comparisons of exact algorithms. More challenging are the EUCLID CALMA, the COST 259, and the ROADEF Challenge 2001 instances whose size varies from 20 to 5000 vertices. Approximate algorithms are usually tested on these instances. Several publications studied the application of SLS algorithms to the FAP problem considering different minimisation objectives. The most relevant publications on construction heuristics, and metaheuristics are those of Hao et al. (1998), Capone and Trubian (1999), Hurley et al. (1997), Castelino et al. (1996), Kendall and Mohamad (2004), and Smith et al. (2002).

The motivation of our study is mainly that of providing a clear comparison of approximate algorithms on challenging and precisely characterised instances for graph colouring generalisation problems. In publications concerning frequency assignment, we often read sentences like "any attempt to evaluate the relative performance of existing algorithms is certain to prove difficult, and this difficulty tends to increase rather than decrease as more papers are published" (Smith et al., 2002). Similar assertions can be found in other publications. Ultimately, they leave the reader confused about which algorithm

[3] DIMACS/DIMATIA/Renyi Working Group on Graph Colourings and their Generalizations. January 2005. http://dimacs.rutgers.edu/Workshops/GraphColor/main.html (March 2005)

[4] A. Eisenblätter and A. Koster. June 2000. <http://fap.zib.de/biblio/> (March 2005)

should be implemented. Even publications in the context of the DIMACS "Computational Challenge" are misleading because of missing details (Lim et al., 2003) or because algorithms have never been directly compared due to results attained under excessive long run times (see for example Dorne and Hao, 1998b) or different instances. The methods described in Chapter 3 give us the tools for the design and analysis of experiments to carry out a correct comparison of algorithms and to address precise questions on the solution approaches.

The focus of our study is on a special case of the set T-colouring problem, namely the *separation distance set T-colouring* problem. Furthermore, given that the majority of the publications have focused on the minimisation of the span, we also address mainly this objective. However, we will also try to understand whether the evaluation of solutions that approximate the minimal span remains the same when considered in the light of the minimal order criterion.

5.4. Benchmark instances

Instances for computational benchmarking of solution methods for graph colouring generalisations are collected in the context of the "DIMACS Computational Challenge on graph colouring and its generalisations" and they are available on the web.[5] Instances can be grouped by problem:

Precolouring extension. Quasigroup instances with precoloured vertices. They are obtained by uncolouring randomly assignments from a feasible colouring (from there the name Quasigroup with "holes"). The instances can be solved both by finding a feasible k-extension for a given k with $\Gamma = \{1, \ldots, k\}$ or by minimising the order of the extension. We will focus only on the former that corresponds to the decision version of the problem. Instances are denoted by `qwhdec`.

Set T-colouring. Two types of random graphs are given. By ignoring some data, these instances can be used to study the GCP as well as the T-colouring problem. The focus is on minimising the span of the colouring, which corresponds to minimising k with $\Gamma = \{1, \ldots, k\}$ such that a feasible colouring exists.

Random Graphs. These instances were introduced by Dorne and Hao (1998b), although their use remained limited to that publication. These are based on Uniform random graphs with n vertices and an edge probability p, expressed in percent (see also page 90). Requirements, vertex-distances and edge-distances are assigned to vertices and edges by uniformly choosing in the intervals $\{1, \ldots, r\}$, $\{1, \ldots, t_{uu}\}$, $\{1, \ldots, t_{uv}\}$, respectively. In the set of instances the following values are considered $n \in \{30, 100, 300, 500, 1000\}$, $p \in \{10, 50, 90\}$, $r = t_{uu} = t_{uv} = 5$. We denote these instances with the original French nomenclature `essai`$.n.R.p$ where $R = \sum r(v)$.

New Random Graphs. For some extensive experiments described in the next sections, solving the previous graphs resulted computationally prohibitive. We therefore modified the generator of Culberson et al. (1995) for graph colouring instances to produce set T-colouring instances. We confine ourselves to

[5] M. Trick. "Computational Series: Graph Coloring and its Generalizations." August 2001. `http://mat.gsia.cmu.edu/COLOR04/`. (July 2005.)

consider Uniform random graphs without hidden colouring but we controlled the requirements at the vertices and the distances constraints. As for the previous class, we vary requirement, vertex-distance, and edge-distance constraints assigned to vertices and edges by uniformly choosing them in the intervals, $\{1,\ldots,r\}$, $\{1,\ldots,t\}$, and $\{1,\ldots,c\}$, respectively. We considered the following set of parameters: $n \in \{60, 120, 240\}$, $p \in \{0.1, 0.5, 0.9\}$, $r \in \{5, 10\}$, $t \in \{5, 10\}$, and $c = t$. We denote these graphs as T-G.r.t-n.p.

Geometric graphs. They are generated by random points in a 10,000 by 10,000 grid and are connected by an edge if they are close enough. Distances associated to edges are inversely proportional to the distances between nodes in the grid while vertex colour requirements are uniformly generated. The size of these instances ranges from 20 to 120. We denote sparse instances as GEOMn and denser instances as GEOMna and GEOMnb. The instances GEOMnb have higher requirements per node than GEOMna.

5.5. Algorithms for the Precolouring Extension Problem

Exact algorithms for solving precolouring extensions have recently been studied by Gomes and Shmoys (2002). Heuristic algorithms for solving this problem are easily derived from those presented for the GCP. Here, we confine ourselves to consider the decision version of this problem consisting in finding a feasible colouring (that certainly exists) for the Quasigroup instances with "holes" solved by Gomes and Shmoys (2002). In particular, we are interested to give insight into the following two issues: (i) which is the best solution approach for tackling the problem and whether the relative performance of the algorithms is the same as for the GCP; (ii) understanding whether the hardness of the instances varies for SLS algorithms as for the exact algorithms studied by Gomes and Shmoys (2002).

5.5.1. Two solution approaches

We consider two basic solution approaches.

Solving the reduced graph. In Section 5.2.8, we described the transformation of a precoloured graph into a reduced graph. Solving the reduced graph can be accomplished by any of the algorithms described in Chapter 4. In particular, we consider construction heuristics with a bound k_b. For the ROS and DSATUR heuristic we modified Algorithm 4.1 on page 97 in such a way that if $l > k_b$ a number is randomly chosen in $\{1,\ldots,k_b\}$. A similar, straightforward modification was adopted for RLF, where the further constraint $i < k_b$ is added to the outer "while" loop of Algorithm 4.2 on page 97 and the vertices left at the end are assigned a colour at random.

We included in the analysis two enhanced versions of DSATUR and RLF, namely, DSATUR$^+$ and RLF$^+$. There are two new features in DSATUR$^+$, namely (i) the colour assigned to a selected vertex is the colour which can still be feasibly assigned to the lowest number of remaining vertices; (ii) if no colour can be assigned without breaking feasibility, then the colour which creates the lowest number of constraint violations is used.

The novelty of RLF$^+$ lays, instead, in the choice of the first vertex to move in C_i (see Algorithm 4.2) which is not anymore the vertex with maximal degree in V' but the vertex with the lowest number of feasible colours (most constrained vertex).

Among the complete SLS methods we consider only TS$_{\mathcal{N}_1}$, given its good results for the GCP.

Solving the original graph. Clearly, it is also possible to solve the original graph and modify the algorithms to take into account for the constraints related with precoloured vertices. For the construction heuristics this is easily achieved by an opportune initialisation of the data structures involved. For Tabu Search, it would be enough to avoid the involvement of precoloured vertices in any exchange.

5.5.2. Experimental analysis

In Figure 5.3, we compare the solutions of construction heuristics before (left) and after the application of Tabu Search (right). Precisely, the results on the right are relative to Tabu Search started from the initial solutions produced by each of the construction heuristics and with a termination criterion of $1000 \cdot |V|$ iterations. The comparison is performed on the 15 Quasigroup instances with holes from the DIMACS repository and it is based on ranks of the number of violations in the final colourings. The different width of the confidence intervals in the figure is determined by the use of a slightly different design in the two cases: 10 runs per instance for the graphs on the left and 3 runs for the one on the right. The results clearly indicate that DSATUR$^+$ is the best construction heuristic. From the perspective of computation time a comparison is not relevant, since all heuristics remain below 1 second despite the fact that some instances reach 4900 vertices. In general, solving the reduced graph helps in achieving better results and this is particularly evident after the application of Tabu Search. Since $|V|$ is much smaller in the reduced graph, Tabu Search not only performs better in this case but it is also much faster.

In Table 5.1, we report the numerical results of the comparison of Figure 5.3 between DSATUR$^+$ and TS$_{\mathcal{N}_1}$ starting from DSATUR$^+$. We divide the instances in under-constrained (U), medium-constrained (M), and critically-constrained (C) as suggested by Gomes and Shmoys (2002). This division corresponds to the proximity of the instance to the region of phase transition, *i.e.*, the region where the problems pass from being satisfiable to being unsatisfiable. The closer to this region the harder the instance become. Gomes and Shmoys (2002) compare on these instances two different encodings for the problem, as a CSP or a SAT problem, and solve them with exact methods. Results are slightly in favour of the CSP-based approach although the solution method is more sophisticated and there are cases where the SAT-based method finds a solution while the CSP-based does not. Computation times reach peaks of 1.5 hours for these approaches. Our results show that SLS methods encounter similar difficulties as the exact approaches described by Gomes and Shmoys (2002) when solving the critically-constrained instances; problems are therefore hard independently of the search strategy adopted. Slightly different is the performance on the medium-constrained instances where SLS methods are very effective for the two instances of size 70 but fail to solve the others on which the methods of Gomes and Shmoys (2002) still succeed. These instances are challenging and may be particularly suitable for inspiring further research for im-

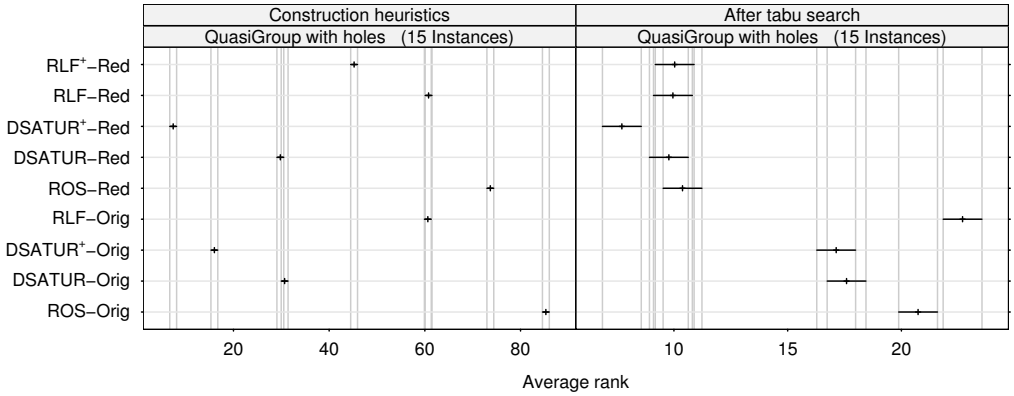


Figure 5.3.: Confidence intervals for the all pairwise comparisons of algorithms on the Quasigroup instances with holes. On the left, the comparison is among construction heuristics obtained collecting 10 runs per instance. On the right, the comparison involves the same construction heuristics and is based on the results after the application of $\mathsf{TS}_{\mathcal{N}_1}$ for $1000 \cdot |V|$ iterations and 3 runs per instance.

Instance	$\|V\|$	$\|V^{red}\|$	% red.	region	DSATUR$^+$ Viol.	DSATUR$^+$ sec.	$\mathsf{TS}_{\mathcal{N}_1}$ Viol.	$\mathsf{TS}_{\mathcal{N}_1}$ sec.	Gomes and Shmoys
qwhdec.o.5.h.10	25	10	60	U	0.0	0.00	0	0.00	CSP/SAT
qwhdec.o.18.h.120	324	120	63	U	5.5	0.00	0	0.03	CSP
qwhdec.o.30.h.316	900	316	65	U	26.0	0.04	3	9.94	CSP
qwhdec.o.30.h.320	900	320	64	U	30.5	0.04	5	10.57	CSP
qwhdec.o.35.h.405	1225	405	67	M	45.0	0.08	22	26.51	CSP
qwhdec.o.40.h.528	1600	528	67	M	47.5	0.16	25	47.92	CSP
qwhdec.o.60.h.1440	3600	1440	60	M	63.5	1.40	24	200.92	SAT
qwhdec.o.60.h.1620	3600	1620	55	M	57.0	1.40	19	179.78	CSP
qwhdec.o.70.h.2450	4900	2450	50	M	55.0	3.55	0	18.75	CSP
qwhdec.o.70.h.2940	4900	2940	40	M	42.5	3.85	0	11.36	CSP
qwhdec.o.33.h.381.bal	1089	381	65	C	49.0	0.07	22	22.49	SAT
qwhdec.o.50.h.825.bal	2500	825	67	C	82.5	0.40	37	148.19	CSP
qwhdec.o.50.h.750.bal	2500	750	70	C	92.0	0.44	43	146.52	N.S.
qwhdec.o.60.h.1080.bal	3600	1080	70	C	122.5	1.35	56	315.84	N.S.
qwhdec.o.60.h.1152.bal	3600	1152	68	C	118.5	1.40	47	289.90	N.S.

Table 5.1.: Numerical results on the precolouring extension instances. Given are the number of vertices and the number of vertices after graph reduction, the percentage of removed vertices, and the region of hardness as defined by Gomes and Shmoys (2002). The results of the two heuristics are expressed in terms of the median number of violations in the final solution and median time limit (or where feasible solutions are found the median time to reach these solutions). For Gomes and Shmoys (2002), we report the approach that found a solution or "Not Solved" in case no solution was found.

proving SLS methods because feasible solutions are known to exist and the hardness of these instances is known *a priori*.

We conclude this section with a final remark on the list colouring problem. All the methods discussed in Chapter 4 can be easily adapted to solve the list colouring problem in a similar vain as we adapted algorithms for the precolouring extension problem. It is enough to add a tabu restriction for each pair colour–vertex which is forbidden by the list constraints.

5.6. Algorithms for the Set T-Colouring Problem

We follow in this study of the set T-colouring problem the same scheme developed in Chapter 5 for the GCP. We first introduce reduction techniques, exact algorithms, and develop a method for gaining lower bounds. Then, we study Stochastic Local Search methods separately in their components and finally analyse the results of a large scale experimental design for the comparison of high-performing algorithms.

5.6.1. Graph reduction

In the GCP, a vertex v can be removed from G if $d(v) < k$ since a feasible colour assignment for v can be found independently from the colours used in $A_V(v)$. A similar reasoning can be applied to the separation distance T-colouring problem. Indeed, each colour chosen for a vertex $w \in A_V(v)$ forbids $2(t_{vw} - 1)$ colours for v. Thus, in total at most $\sum_{w \in A_V(v)} r(w)2(t_{vw} - 1)$ colours can become unavailable for v. If the number of remaining colours available for v is at least $r(v)(t_{vv} - 1) + 1$, we are assured that a feasible colouring for v exists. This removal process can be iteratively applied and the elimination of some vertices can make other vertices removable. However, for our benchmark instances, this reduction is not very effective. For the random graphs no reduction occurs. Only for the Geometric instances some vertices can be removed, but in all cases less than 20%.

5.6.2. Exhaustive search

The goal of this section is showing that the computation cost for solving exactly the set T-colouring problem increases considerably with respect to the GCP and that the exact approach is infeasible already on small graphs. We show this by showing that it is already very hard to solve the *simplification* of a set T-colouring problem into a T-colouring problem. Every instance of set T-colouring problem can be simplified into an instance of T-colouring problem by neglecting the constraints at the vertices, *i.e.*, the vertex requirements and the vertex distance constraints, and maintaining only the original graph and the edge distance constraints. Note that this process of *simplification* is different from the one of *transformation* introduced in Section 5.2.8, according to which an instance of set T-colouring problem can be solved by solving a T-colouring problem on the split graph.

Given this premise, we focus our attention on exact algorithms for the T-colouring problem. Exact approaches for the T-colouring problem are mainly based on backtracking techniques. Examples are given by Hurley et al. (1997) and Costa (1993); Lanfear (1989) considers only the special case of the two-level T-colouring problem.

We modified Ex-DSATUR of Mehrotra and Trick (1996), described in Section 4.6, resulting in a generalised Ex-DSATUR, G-Ex-DSATUR for short. Unfortunately, here a clique cannot be used to precolour a part of the graph as in the GCP. In the next section, we will present a lower bounding technique for deriving from a clique a lower bound for the T-colouring problem. This lower bound is already used by G-Ex-DSATUR for stopping the search, although not for determining a precolouring of part of the graph. Empirical observations gave evidence that there is no impact either in the use of the lower bound and

in the use of the corresponding precolouring. An upper bound is also used for pruning the search when opportune and it is determined by one of the construction heuristics, which will be introduced later. Besides this, G-Ex-DSATUR works as usual by colouring a vertex v with colour c, and contextually forbidding by a forward mechanism the colours in the interval $\{c - t_{u,v}, c + t_{u,v}\}$ for all adjacent vertices $u \in A_V(v)$. Vertices are considered sequentially with precedence given to vertices with the largest number of forbidden colours. If no colour lower than the upper bound can be determined, then the vertices adjacent to the current vertex are recoloured through backtracking until a feasible T-colouring can be established. Ties, in case of an equal number of colours in adjacent vertices, are broken by the vertex degree. We also investigated the use of an adjusted vertex degree which accounts for distances on the edges but it did not yield any improvement.

In Table 5.2, we present results on the instances of set T-colouring problem when *simplified* into T-colouring, and when *transformed* into set T-colouring instances. For both cases we solve the GCP on the corresponding graph with Ex-DSATUR. The increase in the hardness of the T-colouring problems in comparison with the GCP is evident. Whereas Ex-DSATUR is able to solve easily the Geometric instances, both for the original graph and the split graph, G-Ex-DSATUR, encounters enormous difficulties already at very small graph sizes and already on the simplified T-colouring problem; very rarely it returns a solution. For reference purposes, we report in the table also the lower bound and the results of a construction heuristic, which we will introduce later.

5.6.3. Lower bounds for the minimal span

From the previous results it is clear that exact methods are strongly limited in their ability to solve the set T-colouring problem and that the application of approximate methods is required. Note that in the field of frequency assignment it is possible to encounter instances of size even greater than 5000 vertices (Aardal et al., 2001) and that the allocation of frequencies can be required to be very fast if the demand for channels at the transmitters varies over time, as in dynamic environments (Eisenblätter et al., 2002). Hence, the problem to solve is even harder than those reported in Table 5.2.

Approximate methods exhibit the known flaw that they do not provide any guarantee on the quality of the solutions attained. The availability of lower bounds may help in the assessment of their performance. For the set T-colouring problem, an easily derived lower bound is given by the vertex constraints. Accordingly, it is $sp \geq \max_{v \in V}\{r(v)(t_{vv} - 1) + 1\}$. As for the GCP, another lower bound can be derived by the size of a clique. This bound can be generalised to the T-colouring problem. If all the edges contained in a clique of size k have the same forbidden distance t, then the T-span $sp_T(G)$ is at least $t \cdot (|\omega(G)| - 1)$. However, the best lower bound does not necessarily belong to the largest clique and if the distances are not all equal, other bounding procedures must be devised.

Raychaudhuri (1994) observes that, given an assignment of colours on a graph G, vertices can be ordered in a non-decreasing sequence of colours. Then, if the graph is extended to a complete graph G' by introducing edges with distance $t = 0$, the order of the vertices forms a path with length less or equal to the span of the assignment. Hence, the minimum Hamiltonian path in an arbitrary subgraph (completed by zero-value edges) provides a lower bound on the minimal span for the T-colouring problem defined on that subgraph, and thus, on the whole graph. The minimal length Hamiltonian path of

	set T-colouring *simplified* into T-colouring						set T-colouring *transformed* into T-colouring					
	Ex-DSATUR			G-Ex-DSATUR		G-DSATUR	Ex-DSATUR			G-Ex-DSATUR		G-DSATUR
Instance	$\chi(G)$	sec.	LB	k	sec.	k	$\chi(G_S)$	sec.	LB	k	sec.	k
GEOM60	6	0.00	26	-	-	33/34	(37)	43200	230	-	-	258/258
GEOM70	8	0.00	27	-	-	38/40	44	0.00	260	-	-	283/287.5
GEOM80	8	0.00	40	-	-	44/45	63	0.00	365	-	-	392/395
GEOM90	8	0.00	45	-	-	46/46	51	0.00	313	-	-	335/338.5
GEOM100	9	0.00	49	-	-	53/54.5	60	0.00	378	-	-	412/416
GEOM110	9	0.00	49	-	-	53/56.5	62	0.10	348	-	-	400/410
GEOM120	11	0.00	58	-	-	61/66	(64)	43200	343	-	-	412/419
GEOM30a	6	0.00	26	27	18.97	29/31	40	0.00	182	-	-	238/238
GEOM40a	7	0.00	26	-	-	38/38	46	0.00	160	-	-	229/229
GEOM50a	9	0.00	30	-	-	53/54	61	0.00	199	-	-	335/345
GEOM60a	10	0.00	37	-	-	54/55	65	0.00	290	-	-	369/373
GEOM70a	11	0.00	55	-	-	68/69	71	0.00	425	-	-	487/487
GEOM80a	12	0.00	45	-	-	70/70	68	0.00	241	-	-	388/396
GEOM90a	12	0.00	55	-	-	71/76	65	0.1	285	-	-	398/405
GEOM100a	13	0.00	57	-	-	73/78	81	0.1	302	-	-	462/471
GEOM110a	14	0.00	63	-	-	79/81	14	0.0	385	-	-	523/523
GEOM120a	16	0.00	78	-	-	92/93	93	0.2	514	-	-	571/578
GEOM20b	3	0.00	12	13	6.36	14/14	8	0.00	39	-	-	45/45
GEOM30b	5	0.00	14	(26)	12h	26/26	11	0.00	38	-	-	78/78
GEOM40b	7	0.00	30	-	-	35/35	14	0.00	74	-	-	79/86
GEOM50b	8	0.00	32	-	-	37/40	17	0.00	67	-	-	92/94
GEOM60b	9	0.00	37	-	-	47/47	22	0.00	79	-	-	123/132
GEOM70b	10	0.00	40	-	-	55/55	22	0.00	94	-	-	133/135
GEOM80b	12	0.00	51	-	-	69/74	(26)	43200	110	-	-	148/149
GEOM90b	15	0.00	58	-	-	82/83.5	28	0.00	112	-	-	160/161
GEOM100b	15	0.00	59	-	-	82/87	30	0.00	133	-	-	173/179
GEOM110b	15	0.00	68	-	-	89/89.5	37	0.00	182	-	-	221/225.5
GEOM120b	16	0.00	73	-	-	96/101.5	34	0.00	172	-	-	206/219.5
essai.100.275.10	5	0.00	11	-	-	19/20	(17)	43200	46	-	-	61/61
essai.300.937.10	-	-	11	-	-	38/40	-	-	35	-	-	110/111.5
essai.500.1507.10	-	-	13	-	-	53/55	-	-	42	-	-	140/144
essai.1000.3049.10	-	-	15	-	-	86/90	-	-	43	-	-	220/225
essai.30.95.50	7	0.00	9	19	0.43	21/23.5	(24)	7200	31	-	-	68/68
essai.100.304.50	(15)	43200.0	17	-	-	53/53.5	(55)	43200	55	-	-	137/142.5
essai.300.905.50	-	-	19	-	-	124/124	-	-	63	-	-	330/333
essai.500.1484.50	-	-	21	-	-	188/190	-	-	65	-	-	470/480
essai.1000.3024.50	-	-	20	-	-	334/336.5	-	-	74	-	-	862/869
essai.30.90.90	15	0.00	19	34	197.10	40/40	52	0.02	74	-	-	119/119
essai.100.299.90	37	128.00	32	-	-	94/97	(134)	43200	103	-	-	259/259
essai.300.940.90	-	-	46	-	-	239/244.5	-	-	169	-	-	659/669
essai.500.1536.90	-	-	53	-	-	369/375	-	-	197	-	-	1013/1028
essai.1000.2975.90	-	-	62	-	-	673/679	-	-	218	-	-	1804/1821.5

Table 5.2.: Results for the Geometric and Uniform random graphs considered as T-colouring and as set T-colouring problems. G-Ex-DSATUR is compared with a lower and an upper bound. The lower bound LB is computed through Algorithm 5.1 and the upper bound through the construction heuristic G-DSATUR, both introduced in the next sections. For reference, we also report the result of Ex-DSATUR which solves the GCP finding the chromatic number χ of the original and split graph. Both G-Ex-DSATUR and Ex-DSATUR were run with a time bound of 12 hours (43200 sec.). Results in parenthesis indicate that at that time the algorithm did not yet finish. All results in the table are the number of colours used by the solution, yielding a span of $sp^T(G) = k - 1$.

a graph can be determined by solving the minimum Hamiltonian cycle problem (better known as the Travelling Salesman Problem, for short TSP) on the same graph and removing the longest edge.

This bounding procedure has the pitfall that its quality strongly depends on the chosen subgraph. Induced subgraphs of G can provide better lower bounds than G, as shown in Figure 5.4. Janssen and Kilakos (1999) suggest to compute a lower bound for sp_T by finding a lower bound for the TSP in a limited number of "dense" subgraphs of G.[6] We generalise this procedure by solving exactly the TSP problem in large cliques of G. Our procedure is formalised in Algorithm 5.1. Finding large cliques in G is achieved by the variant of the semi-exhaustive greedy procedure of Johnson et al. (1991), as already pointed out in Section 4.7,[7] while the travelling salesman problem is solved with the Con-

[6] Janssen and Kilakos (1999) use this procedure to prove the optimality of the instance Philadelphia, nevertheless, no methodology for determining the "dense" subgraph is given, and they restrict themselves to consider a known clique of the instance.

[7] The maximal clique corresponds to the maximal independent set in the complement graph. The procedure of Johnson et al. (1991) available on the Internet solves the maximal independent set problem in the same way as the XRLF algorithm described in Section 4.10.3 with `EXACTLIM`$= 0$. Yet, the available procedure

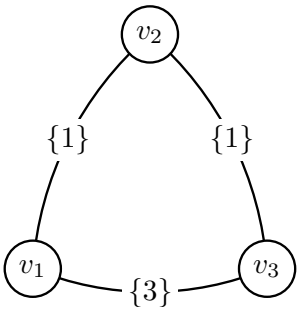


Figure 5.4.: The values on the edges represent the distances T_{uv}. The graph G is complete. A Hamiltonian path of minimal length in G is v_1, v_2, v_3 and has length 2. Hence, the lower bound provided by solving a TSP problem is 2. However, the edge (v_1, v_2) provides a better lower bound of 3 (which, in this case, is also the optimal solution).

```
Function T-Span_Lower_Bound(G, T);
lwb = 0; W^b = ∅;
for i=1 to ITERATIONS do
    Select some large clique W ∈ G;
    if |W| ≥ |W^b| then
        π = an Hamiltonian cycle of minimal length in W;
        p = the Hamiltonian path obtained from π by removing the edge
        (π(i), π(i + 1)) with longest distance t_{π(i)π(i+1)};
        if lwb < c(p)  %c(p) is the total cost of the path.
        then
            lwb = c(p);
            W^b = W;
        end
    end
end
return lwb;
```

Algorithm 5.1: Lower bound procedure for T-colouring. The parameter ITERATIONS is set to 100 for graphs G with less than 1000 vertices, and to 50 for larger graphs.

corde TSP solver (Applegate et al., 1998) which produces an exact solution.[8] Concorde is very fast in solving even large TSP instances, hence this approach is perfectly feasible for our benchmark instances.

Besides lower bounds, also approximation measures may be useful. Dorne and Hao (1998b) propose an approximation value $\widehat{sp}_T(G)$ for the T-span in the case of random graphs $G_{n,p}$ with edge distances t_{uv} randomly chosen from a uniform distribution with mean value $\bar{t}$:[9]

$$\widehat{sp_T}(G) = (\widehat{\chi}(G) - 1) \cdot \bar{t}$$

does not minimise the number of edges in the remaining graph. Therefore, we had to re-implemented it in the case of XRLF. The other parameters required by the algorithm are set as follows: TRIALNUM $= 4$; CANDNUM $= |V|/4.5$; and SETLIM $= \{80, 50, 30\}$ for $\rho(G) \leq 0.5$, $0.5 < \rho(G) \leq 0.8$, and $\rho(G) > 0.8$, respectively.

[8] Concorde TSP uses QSopt linear programming solver. They are both available, for research purposes, at http://www.tsp.gatech.edu/concorde.html. (June 2005.)

[9] We use the hat to indicate that the value is an approximation as done for the GCP.

Function T-greedy($G(V,E), \pi, T$);
$\Gamma = Z^+ \setminus 0$;
$\varphi(v_{\pi(1)}) = \{1\}, \varphi(v_{\pi(2)}) = \emptyset, \ldots, \varphi(v_{\pi(n)}) = \emptyset$;
for $i = 2, \ldots, |V|$ **do**
  $\varphi(v_{\pi(i)}) = \min\{k \in \Gamma : \forall j < i, |\varphi(v_{\pi(i)}) - \varphi(v_{\pi(j)})| \notin T, (v_{\pi(i)}, v_{\pi(j)}) \in E\}$;
end
Let $k = \max\{h : \varphi(v_i), v_i \in V\}$;
return the k-T-colouring, *i.e.*, k and $\varphi(v_1), \varphi(v_2), \ldots, \varphi(v_n)$;

Algorithm 5.2: Greedy construction heuristic for T-colouring.

A similar result is given for the set T-colouring problem. In particular, if the requirements for each vertex are randomly chosen from a distribution with mean value $\bar{r}$, in case of graphs with low edge density, it is:

$$\widehat{sp}_T^s(G) = (\widehat{\chi}(G_{n,p}) - 1) \cdot \bar{t} \cdot \bar{r},$$

while for graphs of high edge density

$$\widehat{sp}_T^s(G) = (\widehat{\chi}(G_{n,p}^S) - 1) \cdot \bar{t}$$

where G_S is the split graph defined in Section 5.2.8.

5.6.4. Construction heuristics

We discuss first three construction heuristics for the T-colouring problem. The best of them is then compared against construction heuristics specific for the set T-colouring.

Greedy T-colouring algorithm

As for the graph colouring problem, there is a basic algorithm for constructing solutions to the T-colouring, the *greedy heuristic*. It was first studied by Cozzens and Roberts (1982). The algorithm recursively colours the vertices of the graph with the smallest feasible colour, proceeding in some defined vertex order. It can be adapted to the set T-colouring problem by assigning a set of feasible colours, instead of a single one, and by choosing the colours for the set as those corresponding to the smallest feasible integers. The algorithm always produces a feasible colouring for any graph. Theoretical studies showed that for some specific graphs and some specific sets T the algorithm finds a colouring with minimal span. We refer to Murphey et al. (1999) for a review of these results on complete graphs.

Algorithm 5.2 gives the greedy heuristic, from now on called T-greedy. It takes as input a graph G and a permutation π of the vertices. A fast implementation of the T-greedy algorithm maintains a set of forbidden colours for each vertex that is still to colour and updates it at each iteration of the algorithm. The final complexity is $\mathcal{O}(k|V|)$. Note that, differently from graph colouring, k is not bounded by the number of vertices.

Clearly, the quality of the solution found depends on the permutation π. We analyse static and dynamic ways to determine this order.

Sequential heuristics for T-Colouring

The greedy algorithm is used jointly with a procedure for determining the order of the sequence of vertices to be coloured incrementally. In the static approach, all vertices are ordered at once, according to a prescribed strategy and then they are coloured in this order. Inspired by the vertex degree information in graph colouring, the strategies are the following:

1. Random vertex ordering (RO algorithm).
2. Largest first vertex ordering (LF algorithm).
3. Smallest last vertex ordering (SL algorithm).

The LF algorithm orders vertices in non decreasing order of their degree. The SL algorithm arranges the vertices in an order $v_1, v_2, \ldots, v_n$ such that vertex v_i has the smallest degree in the subgraph $G' \subset G$ induced by $V' = \{v_1, v_2, \ldots, v_i\}$. This sequence is determined starting at the final vertex and proceeding in reverse order. We denote the three heuristics G-ROS, G-LFS, and G-SLS.

Certainly, the vertex degree still remains a valid heuristic information to determine the hardness of colouring a vertex. However, given the distance constraints of T-colouring it is reasonable to test an *adjusted vertex degree* that uses the additional information of the distance of colours between adjacent vertices and that is defined as $d^T(v) = \sum_{u \in A_V(v)} t_{uv}$. We consider therefore the defined heuristics in the two variants, with vertex degree and with adjusted vertex degree.

A clear comparison of sequential heuristics for T-colouring is not reported in the literature. Hurley et al. (1997) conclude that it is difficult to assess which variant of the sequential algorithm performs well on a specific instance; however, their comparison is limited to small size instances of around 20 vertices and no statistical analysis of the results is applied.

Generalised DSATUR

In the dynamic approach used by DSATUR, the definition of the saturation number remains the same. The only difference occurs in its update. If a vertex v receives colour c then an adjacent vertex u can not receive colours in the interval $\{c - t_{u,v}, c + t_{u,v}\}$.

We tested two different sorting strategies:

1. Largest saturation first vertex ordering (LS algorithm).
2. Smallest saturation first vertex ordering (SS algorithm).

In both cases, ties are broken by the largest adjusted vertex degree. Ties occur, however, much less frequently than in DSATUR for GCP, and this choice has low impact.

In addition to the generalised DSATUR, denoted as G-DSATUR, which uses the T-greedy heuristic to determine the colours to assign to a vertex, we also test a different heuristic that selects the colour among the feasible ones that is also feasible for the lowest number of uncoloured vertices. The algorithm for this greedy procedure, to use in place of the T-greedy heuristic, is given in Algorithm 5.3. We denote with G-DSATUR$^{\#}$ the heuristic that uses the greedy algorithm T-greedy$^{\#}$. In Algorithm 5.3, the case $\Gamma'' \neq \emptyset$, *i.e.*,

Function T-greedy$^{\#}(G(V,E),\pi,T)$;
$\Gamma = Z^+ \setminus 0$;
$\varphi(v_{\pi(1)}) = \{1\}, \varphi(v_{\pi(2)}) = \emptyset, \ldots, \varphi(v_{\pi(n)}) = \emptyset$;
for $i = 2, \ldots, |V|$ **do**
  $\Gamma'' = \{l \in \Gamma : \forall j < i, |l - \varphi(v_{\pi(j)})| \notin T, (v_{\pi(i)}, v_{\pi(j)}) \in E\}$;
    $\Gamma'' \subseteq \Gamma$ is the set of
  %feasible colours $v_{\pi(i)}$
  if $\Gamma'' \neq \emptyset$ **then**
    $\Gamma''' = \{l \in \Gamma'' : \min |\{v_{\pi(j)}, \forall j > i, |l - \varphi(v_{\pi(j)})| \notin T, (v_{\pi(i)}, v_{\pi(j)}) \in E\}|$;
    $\varphi(v_{\pi(i)}) = \min\{l \in \Gamma'''\}$;
  end
  else
    $\varphi(v_{\pi(i)}) = \min\{k \in \Gamma : \forall j < i, |\varphi(v_{\pi(i)}) - \varphi(v_{\pi(j)})| \notin T, (v_{\pi(i)}, v_{\pi(j)}) \in E\}$;
  end
end
Let $k = \max\{h : \varphi(v_i), v_i \in V\}$;
return the k-T-colouring, *i.e.*, k and $\varphi(v_1), \varphi(v_2), \ldots, \varphi(v_n)$;

Algorithm 5.3: A modification of the T-greedy heuristic which selects the least usable colour.

the case in which a vertex can be coloured with a previously used colour, is less likely to occur with dense graphs. However, even for graphs with $\rho(G) = 0.9$ it occurs for at least 50% of the vertices and hence the presence of an effect of the new rule is guaranteed.

The use of saturation degree information on construction heuristics for T-colouring has already been studied by Costa (1993). The method has also been studied for frequency assignment problems by Borndörfer et al. (1998) who analyse a different rule to assign colours (frequencies) and by Hurley et al. (1997) who suggest to colour first vertices in the most constrained path indicated by the lower bounding procedure. Since we did not observe any improvement by this procedure we avoid to use it here.

Generalised RLF

Construction heuristics based on the greedy algorithm try to assign to each vertex the lowest possible colour. In the GCP, we saw that a different strategy is possible: colouring as many vertices as feasible with the same colour before using a new one. In the GCP, this coincides with selecting large independent sets and it is realised by the RLF heuristic. In T-colouring, the concept of independent set is still valid but vertices are also subject to distance constraints and hence it is not enough to assign them to different sets but these sets must also be correctly separated.

The RLF procedure for T-colouring, G-RLF, is given in Algorithm 5.4. The procedure is very similar to Algorithm 4.2 on page 98. The only difference is in the initialisation of the set U of vertices that cannot be added to colour class C. Uncoloured vertices can be added to the new colour class only if no distance constraint is violated. We test two versions of G-RLF, one uses the simple vertex degree and another uses the adjusted vertex degree d^T.

To our knowledge, we are the first to study the extension of the RLF heuristic to the T-colouring problem.

Function G-RLF(G, T);
$C_1 = \emptyset, C_2 = \emptyset, \ldots, C_k = \emptyset$;
$U = V$, $i = 1$; % U is the set of uncoloured vertices
while $U \neq \emptyset$ **do**
  $V' = \{u \in U : \forall j, w \;\; j < i, w \in C_j, (u, w) \in E, i - j \notin T_{uw}\}$;
  $U = U \setminus V'$;
  Let $v \in V'$ be a vertex for which $|d^T_{V'}(v)|$ is maximal;
  %$d^T_{V'}(v) = \sum_{u \in A_{V'}(v)} t_{uv}$
  $C_i = \{v\}$;
  $U = A_{V'}(v)$, $V' = V' \setminus A_{V'}(v)$; % U set of uncoloured vertices which can not be placed in V
  while $V' \neq \emptyset$ **do**
    Let $v \in V'$ be a vertex for which $|d^T_{V'}(v)|$ is maximal; % the edges in U are minimised
    $C_i = C_i \cup \{v\}$;
    $U = U \cup A_{V'}(v)$, $V' = V' \setminus A_{V'}(v)$;
  end
  $i = i + 1$;
end
$k = i - 1$;
return the k-colouring, *i.e.*, k and $C_1, \ldots, C_k$

Algorithm 5.4: Generalised Recursive Largest First heuristic for T-colouring.

Comparison

We evaluate the heuristics according to their capability of minimising the span of the T-colouring. In order to simplify the presentation of the results, we first analyse the effect of the adjustment of the vertex degree. From Figure 5.5 we see that the adjusted information helps G-DSATUR and the sequential heuristics to attain better results while it remains mainly indifferent for the G-RLF heuristic. We focus therefore on the versions that use the adjusted information.

In Figure 5.6, we show confidence intervals for all-pairwise comparisons of the construction heuristics on the various instance classes in a rank-based analysis. G-DSATUR is significantly better than G-RLF and all generalised sequential heuristics. Differently from the GCP, G-RLF is never better and it is not dominated only on sparse Geometric graphs. The use of T-greedy or T-greedy$^{\#}$ in G-DSATUR does not yield significant differences. With respect to time, G-RLF is the slowest heuristic, although the instances when treated as T-colouring, are solved all within 1.5 seconds. G-DSATUR remains below 0.5 seconds on the instances of size 1000.

In order to solve the tie between the two version of G-DSATUR, we investigate how heuristics optimise the use of colours, that is, how they minimise the order of the T-colouring. Note that it is not clear whether the presented heuristics tend to minimise the span or the order. We report the analysis for this criterion in Figure 5.7. Apparently, G-DSATUR is the best in optimising also the order but only on the random graphs. On the Geometric graphs, instead, G-RLF is the best algorithm. The relevance of this result is twofold: on the one hand, it tells us that on the Geometric instances there is a trade off between minimal span and minimal order and that G-RLF minimises better the order; on the other hand, it unveils an important aspect whose reasons would be worthy to be investigated deeper: Uniform graphs and Geometric graphs exhibit different span–

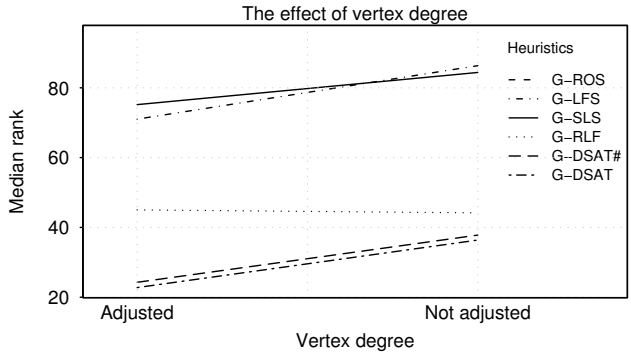


Figure 5.5.: The effect of the adjustment of the vertex degree information on the heuristics. Data are aggregated over all Uniform and Geometric random T-colouring instances and ranked within instances. The number of runs per instance is 10, hence, ranks fall in the interval $[1, 160]$.

order trade offs. Unfortunately, our original intent of breaking the tie in G-DSATUR is not achieved. Therefore, giving preference to simplicity, in the rest of the chapter we will use G-DSATUR with T-greedy heuristic.

Construction heuristics for Set T-Colouring

The construction heuristics for the T-colouring problem can be used also for the set T-colouring when the split graph is considered. However, it is interesting to investigate whether there exist approaches that exploit better the structure of the set T-colouring problem.

Sivarajan et al. (1989) study sequential heuristics specialised for the channel assignment problem. The methods are interesting but the analysis is confined to instances of size 21 and results are not compared with any other approach. Inspired by their work, we considered the following four factors, with two choices for each of them, for the assemblage of a sequential construction heuristic.

Vertex degree adjustment: The alternatives are:

- $d_A^T(v) = \sum_{u \in A_V(V)} r(v)t_{uv} - t_{vv}$. This is the original measure proposed by Sivarajan et al. (1989) (identifier A);
- $d_B^T(v) = \frac{1}{2}r(v)(r(v)-1)t_{vv} + \sum_{u \in A_V(V)}(r(u)r(v)t_{uv})$. This is our alternative proposal which counts the number of individual constraints acting on each requirement $r(v)$, weighted by the number of colours that each constraint forbids, *i.e.*, t_{uv} or t_{vv} (identifier B).

Vertex ordering: based on the degree information, the alternatives are:

- smallest last ordering (identifier C);
- largest first ordering (identifier D).

Requirement order: Once vertices have been ordered, requirements can be arranged in a matrix $|V| \times \max_{v \in V} r(v)$. Each row of the matrix corresponds to a vertex while

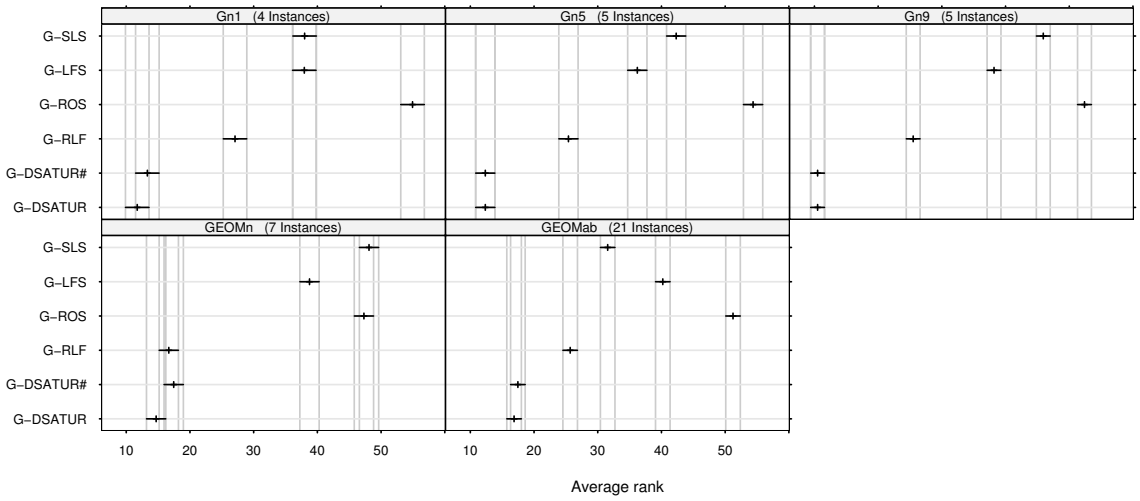


Figure 5.6.: All-pairwise comparisons in a rank-based analysis using the *minimal span* criterion. Only T-colouring instances are considered in a design with 10 runs per pair algorithm–instance. Instances `GEOM`n`a` and `GEOM`n`b` are aggregated in the class `GEOMab` because from a T-colouring perspective their features coincide. The edge density of instances GEOMn is 0.1, and of GEOMab is 0.2, while in both classes the average distance constraint at the edges is 4.5.

each column to a single colour requirement. The rows are arranged in the vertex order determined previously while the colour requirements are arranged such that all the columns of the matrix have nearly the same number of requirements. Requirements in the first row start at the first column. Requirements in the second row start at column $(r(v_1) + 1)$ and fill the row cyclically. Similarly, requirements in the remaining rows start where requirements in the previous end. Once requirements have been arranged in this way in the matrix, two orderings may be obtained:

- row-wise ordering (identifier R:);
- column-wise ordering (identifier C:).

Assignment strategy: Decided the requirement order, two strategies can be adopted to assign the colour:

- Colour exhaustive: assign the least colour that does not introduce any violation to each requirement selected in the defined order.
- Requirement exhaustive: for each colour in order from 1 to k scan all yet uncoloured requirements in the defined order and assign the current colour if feasible.

We tested all 16 possible combinations of levels of these factors on the available set T-colouring instances. In Figure 5.8 we explore the effects and interactions of the four factors on a rank-based analysis. The only evident conclusion is that the requirement exhaustive strategy is a better choice with respect to the colour exhaustive strategy. All other three factors seem to interact with each other and it is not possible, from this figure, to understand the significance of their effects.

We proceed therefore considering all-pairwise comparisons of the 8 combinations left after the removal of the colour exhaustive strategy. In the comparison, we include also a

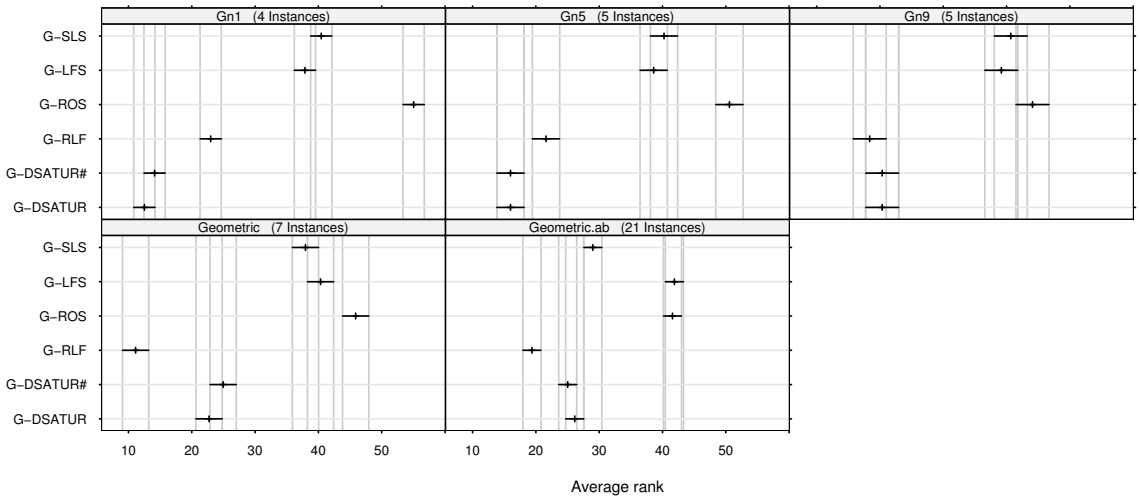


Figure 5.7.: The same experimental design of Figure 5.6 analysed by the *minimal order* criterion.

version of G-DSATUR that solves the set T-colouring problem as a T-colouring problem in the split graph. The results of the comparisons are reported in Figure 5.9 where each of the 8 configurations is indicated by a label obtained by the identifiers of the factor levels. The most relevant observation is that none of the heuristics for set T-colouring is significantly better than G-DSATUR in none of the scenarios. The analysis of the computation time did not reveal any difference among the 9 heuristics included in the comparison. All solve the largest instances of density 0.9, whose split graph consists of 3000 vertices, in about 2.5 seconds, while, for the same size, the computation time falls below 1.5 seconds if the edge density is 0.5. G-DSATUR appears therefore a good choice for solving also set T-colouring instances.

Finally, we examine the criterion of minimising the order of the set T-colouring. In Figure 5.10 the instance classes seem to have a strong influence on the performance of the heuristics with respect to this criterion and no clear winner arises.

5.6.5. Iterative Improvement

There are different approaches for solving the set T-colouring problem by means of local search. Of main importance is understanding whether it is more convenient to solve the set T-colouring by splitting it into a T-colouring problem or to try to exploit the difference between edge and vertex constraints by approaching the problem in its original form. Analogously to the graph colouring problem, it is interesting to investigate the possibility of solving the problem as a sequence of decision problems with k fixed or by varying the number of colours used. We will consider the following three alternatives.

Solving the T-colouring problem on the split graph with k fixed. Once the graph of a set T-colouring problem has been split, the solution representation may be a complete assignment of colours to vertices like in the GCP. The solutions may be represented by a map φ and a partition $\mathcal{C}$ of vertices in colour classes, *i.e.*, $\mathcal{C} = \{C_1, \ldots, C_k\}$. In this way, each vertex receives exactly one colour. The requirement constraints are satisfied

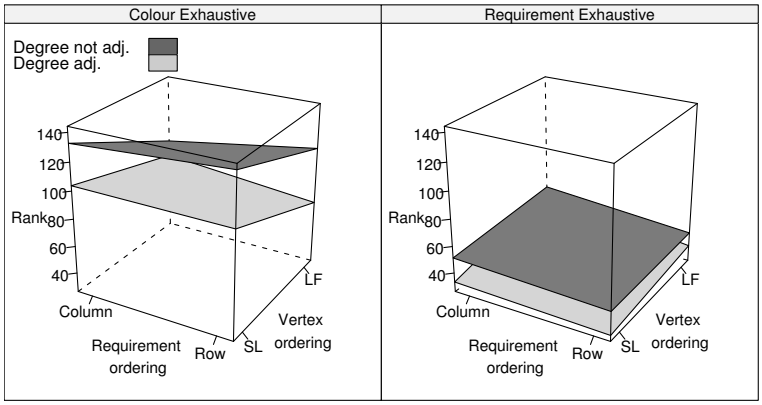


Figure 5.8.: Interaction plots for factors of the set T-colouring heuristics. The four factors are: assignment strategy, vertex degree adjustment, vertex ordering, and requirement ordering.

by the solution representation, while the distinction between vertex constraints and edge constraints is removed, as the former are now on the edges of the split graph. An initial assignment and an initial number of colours is determined by G-DSATUR. The number of colours is then decreased by one by assigning to the vertices which had colour k a colour in $\{1, \ldots, k-1\}$ that introduces the lowest number of edge conflicts, breaking ties randomly. The problem of finding a feasible solution for $k-1$ is solved by a local search minimising the number of edge violations in the solution. The objective function is, similarly to the GCP, defined as

$$f(\mathcal{C}) = \sum_{(u,v)\in E} I(u,v)$$

where $I(u,v) = 1$ if $|\varphi(u) - \varphi(v)| < t_{uv}$, and $I(u,v) = 0$ otherwise. The goal is to find a solution $\mathcal{C}$ such that $f(\mathcal{C}) = 0$.

Solving the T-colouring problem on the split graph with k variable. In this case, the solution representation is the same as in the previous approach but the number of colours may increase or decrease at run time. Local search can visit feasible and infeasible complete assignments that use no more than k_I colours determined by the initial solution produced by G-DSATUR. An evaluation function to guide the search towards feasible colourings and towards colourings with smaller span was proposed by Hurley et al. (1997). It is defined as

$$f(\mathcal{C}) = k_I \cdot \sum_{(u,v)\in E} |I(u,v)| + (k_{max} - k_{min}) + \sum_{i\in\Gamma} J(C_i) + k_{max} \tag{5.1}$$

where $J(C_i) = 1$ if $|C_i| > 0$ and $J(C_i) = 0$ otherwise. Formally, it is possible to express $k_{max} = \arg\max_{i\in\Gamma}\{C_i : J(C_i) = 1\}$ and $k_{min} = \arg\min_{i\in\Gamma}\{C_i : J(C_i) = 1\}$. The function 5.1 computes the sum of edge conflicts, span, order of the colouring, and largest colour used. The edge conflicts are weighted by the largest number of colours, thus a solution which reduces the number of violations will always be preferred with respect to those that modify the other terms of the sum. The inclusion of the order in the sum contributes

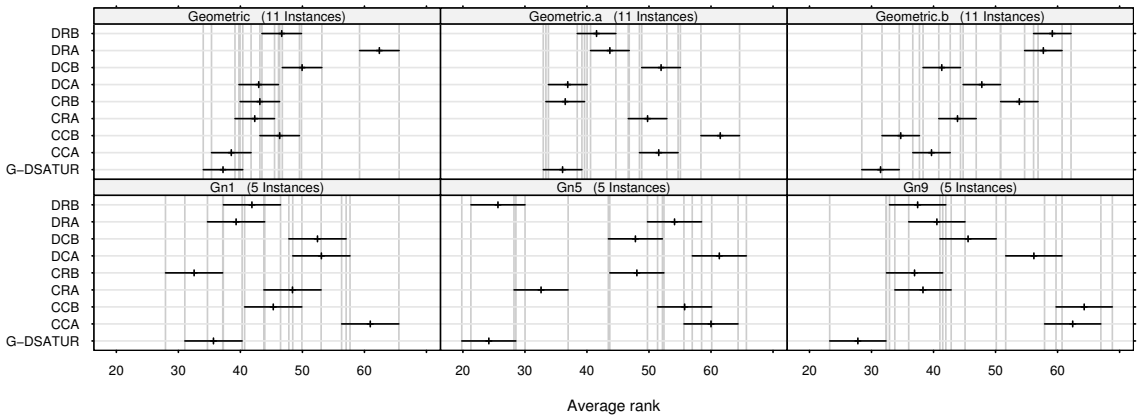


Figure 5.9.: All-pairwise comparisons in a rank-based analysis on the *minimal span* criterion on set T-colouring instances. The label of the heuristics is obtained by adding the identifiers of the factor levels, in the order, vertex ordering, requirement ordering, and vertex degree adjustment.

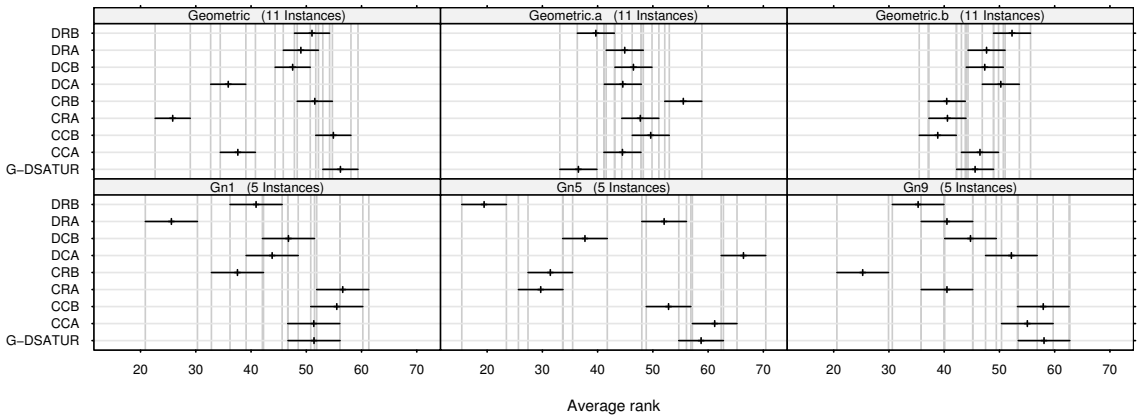


Figure 5.10.: The comparison of Figure 5.9 from the perspective of *minimal order* criterion.

to break ties. The last term is the least important and contributes only to use the first colours, avoiding to move with the same span over and over through the interval $[0, k_I]$.

Solving the set T-colouring problem with k fixed. Maintaining the set T-colouring problem in its original formulation, we can represent a solution as a set of vertices each with $r(v)$ colours assigned. This can be done by creating a vector $\{\varphi(v_1, 1), \ldots, \varphi(v_1, r(v_1)), \ldots, \varphi(v_n, 1), \ldots, \varphi(v_n, r(v_n))\}$ of length $\sum_{v \in V} r(v)$ in which each position corresponds to a colour assigned to a vertex. We continue to denote a solution as φ, although now it is as a multi-valued function, or as $\mathcal{C}$, which is a set of colour classes containing the vertices, *i.e.*, $\mathcal{C} = \{C_1, \ldots, C_k\}$; however, $\mathcal{C}$ is not anymore a partition because vertices can be in several colour classes. The requirement constraints are again satisfied by the solution representation for φ. Thanks to the distinction of positions in the vector, the vertex constraints may also be used to reduce the number of solutions in the search space $\mathcal{S}$ to only those assignments that satisfy vertex constraints. With k fixed, an initial assignment is determined and a problem for finding a feasible colouring is solved

in sequence at each k. The evaluation function counts the number of unsatisfied edge constraints

$$f(\mathcal{C}) = \sum_{(u,v)\in E} \sum_{\substack{i\in\{1,\ldots,r(u)\}\\ j\in\{1,\ldots,r(v)\}}} I(u,v,i,j)$$

where $I(u,v,i,j)$ is 1 if $|\varphi(u,i) - \varphi(v,j)| < t_{uv}$, and 0 otherwise. The goal is to find a solution $\mathcal{C}$ such that $f(\mathcal{C}) = 0$.

Neighbourhood Structure

For the approach to the T-colouring problem we limit ourselves to consider the basic one-exchange neighbourhood $\mathcal{N}_1$ of Definition 4.1 on page 103. For the set T-colouring we consider instead two neighbourhood structures: a modification of the one-exchange neighbourhood and a colour reassignment within vertices.

Definition 5.1 (Restricted one-exchange) The neighbourhood of $\mathcal{C}$ is the set of colourings $\mathcal{C}'$ obtained from $\mathcal{C}$ by changing exactly one colour of one vertex in such a way that the vertex constraints are maintained satisfied. Using the function φ, which defines the set $\{C_1,\ldots,C_n\}$, a formal definition of the neighbourhood is

$$\begin{aligned}\mathcal{N}_1^R(\mathcal{C}) \;=\; & \{\mathcal{C}' : \exists v \in V \text{ and } i \in \{1,\ldots,r(v)\} \text{ and } j \in \Gamma \text{ and } |\varphi(v,l) - j| \notin T_{vv}\ \forall l \neq i\ :\\ & \varphi'(v,i) = j \text{ and}\\ & \varphi'(v,l) = \varphi(v,l)\ \forall l \neq i \text{ and}\\ & \varphi'(u,i) = \varphi(u,i)\ \forall u,i,\ u \in V \setminus \{v\} \text{ and } i \in \{1,\ldots,r(u)\}\}\end{aligned}$$

Definition 5.2 (Intra-vertex reassignment) The neighbourhood $\mathcal{C}$ is the set of colourings $\mathcal{C}'$ obtained from $\mathcal{C}$ by changing the colours assigned to a vertex in such a way that the vertex constraints and the edge constraints involving that vertex are satisfied. In other terms, the application of a move in this neighbourhood produces an *adjustment* of the colours assigned to a vertex such that all its constraints are satisfied. Formally,

$$\begin{aligned}\mathcal{N}_A(\mathcal{C}) \;=\; & \{\mathcal{C}' : \exists v \in V \text{ and } H \subset \Gamma \text{ with } |H| = r(v) \text{ and } |c_1 - c_2| \notin T_{vv}\ \forall c_1,c_2 \in H \text{ and}\\ & |c - \varphi(u,j)| \notin T_{uv}\ \forall (u,v) \in E, c \in H :\\ & \varphi'(v) = H \text{ and}\\ & \varphi'(u,i) = \varphi(u,i) \quad \forall u,i\ u \in V \setminus \{v\} \text{ and } i = \{1,\ldots,r(u)\}\}\end{aligned}$$

From the previous chapter we learnt that very large scale neighbourhoods based on path and cyclic exchanges are not profitable for the graph colouring. Therefore, we avoid to consider them here, since the distance constraints make the neighbourhood exploration even more complex.

Kempe chains are, instead, infeasible in the context of T-colourings, since finding independent colour classes is not enough to satisfy the distance constraints. A generalisation of Kempe chains has been attempted by Borgne (1994). The idea is to select two vertices, or two vector positions, u and v coloured with c_1 and involved in a constraint violation and a colour c_2 whose colour class does not contain any vertex adjacent to u and v. All vertices in C_{c_1}, except u, are then exchanged with all those in C_{c_2}.

```
Function Initialise_Δ(G,φ);
Δ = 0;
for each v in V do
    for each u in A_V(v) do
        for i in {φ(v) − T_uv, ..., φ(v) + T_uv} do
            Δ(v, iφ(v)) = Δ(v, φ(v)) + 1;
        end
    end
end

Function Update_Δ(G,φ,v,C_i) %C_i is the old colour class of v
for each u in A_V(v) do
    for i in {c_old − T_uv, ..., c_old + T_uv} do
        Δ(u, c_old) = Δ(u, i) − 1;
    end
    for i in {φ(v) − T_uv, ..., φ(v) + T_uv do
        Δ(u, i) = Δ(u, i) + 1;
    end
end
```

Algorithm 5.5: Pseudo-code for the initialisation and update of the matrix $\mathbf{\Delta}$.

Remark. Recalling the definition of connectivity of a search space (Definition 3.3, page 80) it is straightforward to show that the search space in the set T-colouring problem solved with k fixed is connected when using the neighbourhood structure $\mathcal{N}_1^R$.

Neighbourhood Examination

One-exchange neighbourhood. The examination of $\mathcal{N}_1$ is done as for the GCP and a complete exploration entails visiting $(k-1)|V|$ possible neighbours, although in practice, only vertices involved in some conflict are considered for exchange.

Additional data structures are used for the fast evaluation of $f(\mathcal{C})$ after an exchange. In the split graph and for fixed k, the matrix $\mathbf{\Delta}$ is defined by $|V| \times k$ elements. Its initialisation and update is outlined in Algorithm 5.5. The complexity of these procedures increases with respect to the GCP and it depends on the entity of the distances between colours. In the case of k variable, the neighbourhood has size $(k_I - 1)|V|$ and the matrix $\mathbf{\Delta}$ is maintained of size $|V| \times k_I$. With the necessary changes, Algorithm 5.5 still applies. In addition, the order of the colouring and the values k_{min} and k_{max} are also maintained. The evaluation function after an exchange for vertex v from colour $\varphi(v)$ to c is $f(\mathcal{C}') = k_I \mathbf{\Delta}(v, c) + \delta_c + \delta_{\varphi(v)}$ where δ_c and $\delta_{\varphi(v)}$ are the contributions of removing v from C_φ and inserting it into C_c, respectively, to the last three components of the sum in Equation 5.1.

Restricted one-exchange neighbourhood. With the set T-colouring representation, a matrix $\mathbf{\Delta}$ of size $\sum_{v \in V} r(v) \times k$ is maintained that indicates for each position in the solution vector corresponding to a requirement the number of edge violations determined by the assignment of a colour to the corresponding vertex. Algorithm 5.5 is still valid for maintaining the matrix although inside the loop on the vertices adjacent to v a further loop has to be included iterating over the requirements of vertex u. Note that in the two previous cases we have $|V| = \sum_{v \in V} r(v)$, while now the set V is simply made of origi-

nal vertices. The complexity of Algorithm 5.5 remains therefore the same. In addition to $\mathbf{\Delta}$, another matrix $\mathbf{\Delta}_2$ of size $|V| \times k$ is maintained to forbid the assignment of colours that break vertex constraints. The update of this matrix after each exchange is done in $\mathcal{O}(2t_{vv})$. The evaluation function is updated after an exchange as $f(\mathcal{C}') = f(\mathcal{C}) + \mathbf{\Delta}(v, c)$.

Intra-vertex reassignment neighbourhood. In practice, a reassignment is attempted only for vertices in conflict. Once a vertex is chosen, a set F is determined comprising the colours which are feasible with respect to the edge constraints acting on the vertex (not the vertex constraints). The construction of this set can be done easily in $\mathcal{O}(|V|k)$ using the same data structure Δ described for the previous neighbourhoods (its update is done by decomposition of the reassignment into single restricted one exchange). If we would simply look for $r(v)$ colours from F such that the vertex constraints are satisfied this would be easy: we just have to order the values in F and scan the set once skipping the values that are not sufficiently distant from the previous ones. Yet, this procedure is deterministic and we would obtain the same reassignment of colours visiting a vertex twice if nothing in its neighbourhood changed. We want to avoid this and make the search randomised, such that, visiting the vertex the second time, we obtain a different configuration that can be profitably propagated. Implementing this strategy corresponds to determine all subsets of F of size $r(v)$ that satisfy the vertex constraints and pick one at random.[10] In our solution to the problem we order randomly the set F and search for a solution by avoiding to order F again. The proposed algorithm is a backtracking approach and it is outlined in Algorithm 5.6. In the algorithm the set H is implemented as an ordered binary tree. This allows the insertion of an element in $\mathcal{O}(\log n)$ operations and to check the vertex constraints in constant time (since only the predecessor and successor are to be considered). The algorithm has exponential worst case, yet we could not find confirmation that the problem is in fact NP$-hard$. In the worst case, Algorithm 5.6 has to consider all possible $\binom{|F|}{r(v)}$ solutions.

In practice, we will never use this neighbourhood alone. In a similar manner as with the VLSN in Chapter 4, we combine it with the restricted one-exchange. How this is done is explained in the next section.

5.6.6. Stochastic Local Search algorithms

As for the GCP, we embed the local search components into more complex SLS methods. We restrict our attention to those methods that performed the best on the GCP.

Solving the T-colouring problem on the split graph with k fixed

Tabu Search, Min-Conflicts, Guided Local Search, and Hybrid Evolutionary algorithm. Mutatis mutandis we reuse the same framework of these metaheuristics based on the one-exchange neighbourhood as described for the GCP (Section 4.10.1). They become therefore generalised versions of these algorithms and we denote them GSF-TS,

[10] More formally, the problem can be stated as
Input: A subset of distinct integers numbers $M \subseteq \{1, \ldots, n\}$, and two integers $c < n$ and $k < |M|$.
Question: Which are all possible subsets $S \subset M$ such that $\forall a, b \in S$: $|a - b| > c$.
Clearly, the problem has no solution if $c < (m - k)/(k - 1)$.

```
Function reassign(G, v, t_vv, r(v));
F = {c ∈ Γ : |c − φ(u,i)| ≥ t_{u,v}, ∀(u,v) ∈ E, ∀φ(u,i) ∈ φ(u)};
if |F| ≥ r(v) then
    H = ∅; shuffle F;
    assign(F, H, t_vv, r(v), 0);
    if (|H| = r(v)) then return an H assignment of feasible colours for v;
end
return no feasible assignment of colours for v found;

Function assign(F, H, d, r, start);
for i ∈ {start, ..., |F| − r + |H|} do
    Let c be the i-th element of F;
    if |c − j| ≥ d ∀j ∈ H then
        assign(F, H ∪ {i}, d, r, i + 1);
    end
end
```

Algorithm 5.6: An algorithm for the exact reassignment of colours to a vertex.

GSF-GLS, GSF-HEA (G stands for generalised, S for split graph, and F for fixed k). As for the GCP, GSF-HEA uses GSF-TS as local search.

In the case of Tabu Search the length of the tabu list is set $tt = \mathit{random}(10) + 2\delta|V^c|$, where V^c is the set of vertices which are involved in at least one conflict, δ is a parameter, and $\mathit{random}(10)$ is an integer random number uniformly distributed in $[0, 10]$.

Note that our GSF-TS is very similar to the Tabu Search methods proposed by Costa (1993), Castelino et al. (1996), and Hurley et al. (1997). In those papers, Tabu Search was shown to perform better than simulated annealing and other genetic algorithms, which, therefore, we avoid to re-implement here. More specifically, we tested the FASoft system Hurley et al. (1997), which however resulted in worse results than GSF-TS, the reason for this being mainly the fact that the starting solution generated by the construction heuristics of that system are always worse than the one obtainable by G-DSATUR. GLS was already applied in the context of frequency assignment by Tsang and Voudouris (1998). Their local search is more complex than ours since it includes the use of *don't look bits* to avoid the repetition of recent exchanges. This difference is, however, not significant, as we also restrict the neighbourhood to vertices involved in conflicts and their method imposes similar restrictions.

Tuning. The tuning of parameters was done separately for Uniform and Geometric instances but priority was given to choices with robust performance. These are the final settings and in parenthesis the alternatives considered:
GSF-TS: $\delta = 20$ $(0.5, 1, 10, 20, 30, 40, 50, 60, 70, 100)$;
GSF-GLS: $\lambda = 1$ and $sw = 20$ $(1, 20, 100, 200)$;
GSF-HEA: Tabu Search iterations 10000 $(1000, 10000)$ and $\delta = 20$ $(20, 40)$;
GSF-MC: $tt = 20$ $(2, 10, 20, 30)$.

Solving the T-colouring problem on the split graph by extending partial graphs

Incomplete Dynamic Backtracking. Prestwich (2002b, 2003) models the T-colouring problem as a binary constraint satisfaction problem and solves it by means of constraint

Let $\mathcal{C}^*$ be the best non tabu neighbour of $\mathcal{C}$ in $\mathcal{N}_1^R$ breaking ties randomly;
if $f(\mathcal{C}^*) < f(\mathcal{C})$ **then return** $\mathcal{C}^*$;
for all $v \in V^c$ of $\mathcal{C}$ **do**
  if reassign$(G, v, t_{vv}, r(v))$ returns an assignment **then**
    return $\mathcal{C}$ with $\varphi(v)$ changed as indicated by reassign
  end
end
return $\mathcal{C}^*$

Algorithm 5.7: The procedure for the selection of a neighbour in GOF-TS-reass. The function reassign is given in Algorithm 5.6.

propagation techniques based on backtracking with forward checking. Nodes in the search tree are feasible partial colourings. The algorithm, called FCNS, ends up to be very similar to the Ex-DSATUR for the GCP but, contrary to that algorithm, it is incomplete to improve scalability. FCNS uses an incomplete randomised form of dynamic backtracking which selects randomly backtrack variables and unassigns them without unassigning those assigned after. In the algorithm, at each dead-end B backtrack variables are unassigned where B is a user defined noise parameter.

Tuning. The code of the algorithm was provided by Prestwich and, according to his suggestion, we set the parameter B to 1.

Solving the T-colouring problem on the split graph with k variable

Tabu Search. We test a Tabu Search algorithm based on the same framework as in the case of k fixed and neighbourhood $\mathcal{N}_1$, but using the evaluation function of Equation 5.1. We denote this algorithm as GSV-TS.

Tuning. We set $\delta = 20$ as for GSF-TS (other values tested in the tuning phase were 0.5, 1, 10, 20, 30, 40, 50, 60, 70, 100).

Solving the set T-colouring problem with k fixed

Tabu Search. We test two Tabu Search algorithms, GOF-TS and GOF-TS-reass. GOF-TS uses the $\mathcal{N}_1^R$ neighbourhood and it is, basically, the same algorithm as introduced by Dorne and Hao (1998b). The version developed for frequency assignment instances by Hao et al. (1998) differs only in the management of the tabu length, while the version of Hao and Perrier (1999) is more rudimentary. GOF-TS-reass is instead a new algorithm that uses the union of $\mathcal{N}_1^R$ and $\mathcal{N}_A$. Since the exploration of $\mathcal{N}_A$ is computationally expensive, a heuristic truncating rule is adopted similar to the Variant 3 for the VLSN in the GCP (see page 113). This rule reduces the exploration of $\mathcal{N}_A$ to the only cases where no improvement can be found in $\mathcal{N}_1^R$. The neighbourhood examination scheme is reported in Algorithm 5.7. Despite this rule, a main difference with $\mathsf{TS}_{\mathsf{VLSN}}$ in the GCP is that the exploration of the neighbourhood remains exponential in the worst case. As we will see, however, for the instances of our analysis its application remains feasible.

A Tabu Search mechanism is used only for restricting $\mathcal{N}_1^R$. For $\mathcal{N}_A$ such a mechanism is not needed, as the randomisation of reassign in Algorithm 5.6 implies already that different assignments are returned at each call of the function and repetitions avoided.

Tuning. For both algorithms GOF-TS and GOF-TS-reass we tuned the parameter δ that influences the tabu length. We set $\delta = 10$ for GOF-TS and $\delta = 20$ for GOF-TS-reass (the other values tested were 0.5, 1, 10, 20, 30, 40, 50, 60, 70, 100)

Adaptive Iterated Construction Search. For the GCP, Culberson (1992) developed an extension of the sequential heuristic based on the greedy algorithm. It consists in applying the greedy method repeatedly each time to a new permutation maintaining vertices that have the same colour in the previous colouring in consecutive positions of the permutation. This method of generating new permutations ensures that the new colouring will not use more colours than the previous colouring. A similar approach can be applied in the T-colouring by using Algorithm 5.2. However, due to the distance constraints, the condition of maintaining vertices that are in the same colour class in consecutive positions of the new permutation is not anymore sufficient to guarantee that the number of colours used will be less or equal the current one. In fact, only permutations in which the distance between colour classes remains the same will guarantee such a property. One such permutation is the reverse order of the colour classes. Lim et al. (2003) propose an algorithm which uses this idea. Nevertheless, details of the implementation are missing and we cannot reproduce exactly the same algorithm. We, therefore, implemented our own version based on the idea of iterated greedy (this implementation yields superior performance to the implementation of Lim et al. (2003), as it arises from the comparison with state-of-the-art results in Appendix C.2). The method works as follows. An initial solution is determined by G-DSATUR. Then, a colouring that uses $k-1$ colours is constructed applying the T-greedy algorithm to a permutation of the vertices obtained from the previous colouring. The permutation maintains vertices that have the same colour in consecutive positions but reverses the order of the colour classes and shuffles vertices in the classes. The rationale behind this procedure is that, if the reverse order of colour classes yields a colouring that uses the same or a lower number of colours, then it is possible that this order, or small variations thereof, lead to a feasible T-colouring with $k-1$ colours. Perturbations in the reverse order are introduced in the permutation by inserting vertices that had colour k before vertices that had colour b, with $b \in \{k-1, \ldots, 1\}$. Each colouring generated by T-greedy on a proposed permutation of vertices is then improved by a Tabu Search local search applied for $I \times V^c \times \sum_{v \in V} r(v)$ iterations. If no feasible solution is found and all possible values of b have been attempted, the search proceeds by Tabu Search solely. We denote with GOF-AG this final algorithm, which is the hybridisation of Tabu Search and Adaptive Iterated Greedy.

Tuning. In the implementation of Lim et al. (2003), b was a parameter to tune and its value was not given. We linked this parameter to the algorithm, and hence we do not need anymore to decide on its specific value. For the parameter I we tested the values 10 and 100 and we noted that $I = 10$ is the best for Uniform graphs while $I = 100$ is the best on Geometric graphs. We maintain this separation. In addition, we run preliminary experiments to decide whether to run the algorithm with GOF-TS or with GSF-TS, as the underlying idea of GOF-AG can be applied to both the original and the split graph. The result was in clear favour of GOF-TS (with its corresponding tuning), hence we include only this version in the comparison.

Solving the T-colouring problem on the split graph by extending partial graphs

Incomplete Dynamic Backtracking. Prestwich (2002b, 2003) attempted to model this problem as a satisfiability problem (SAT) and to solve it with incomplete backtracking techniques for SAT. The number of variables in the model is large and slows down considerably the algorithm yielding inferior solutions to the FCNS method defined above.

5.6.7. Experimental Analysis

In this section, we describe the experimental analysis for the evaluation of SLS methods for solving the set T-colouring instances. In particular, we intend to determine: (i) the best approach to solve the set T-colouring problem (using k fixed versus variable, splitting the graph versus maintain the original distinction of constraints); (ii) the most convenient neighbourhood for Tabu Search; (iii) the best overall algorithm; (iv) the impact of instance features on the comparisons; (v) the scaling of the algorithms with graph size and distance constraints.

In previous literature, comparisons of algorithms for the set T-colouring problem have been accomplished mainly on two classes of instances: the Geometric graphs, available at the DIMACS graph colouring repository, and the random graphs used by Dorne and Hao (1998b). The size of many of these latter graphs resulted excessive for performing a reliable study, therefore, besides Geometric graphs, we use the random graphs generated by ourselves and introduced in Section 5.4.

Analogously to what was done for the GCP, we use CPU time as the termination criterion as a way to ensure that all algorithms do make use of the same computational resources. Other measures, such as number of algorithmic steps, are not possible given that the algorithms are all different and the search steps have different complexity. First, we investigate which could be a reasonable time limit.

The time limit. We use the algorithm GOF-TS as the reference algorithm. This choice is justified by the fact that, as for the GCP, Tabu Search appears to perform well, it is a relatively simple algorithm, and hence it constitutes a reasonable benchmark for more complex algorithms. Furthermore, we will show that its running times are not considerably different from GSF-TS for the same number of iterations, which is another algorithm appealing to serve as reference. Unfortunately, previous comparisons published are incomplete in the definition of the experimental setting adopted (see, for example, Lim et al., 2003). Prestwich (2003) and Dorne and Hao (1998b) define, instead, a maximal number of iterations for their algorithms. Dorne and Hao (1998b) use 10^7 ($2 \cdot 10^7$ for the largest graphs) iterations for their implementation of GOF-TS. We adopt a different approach linking the maximal number of iterations for GOF-TS to the size of the instance, *i.e.*, $I_{max} = 10000 \times \sum_{v \in V} r(v)$. On graphs of size 100 this value yields about one third of the number of iterations used by Dorne and Hao (1998b), while on larger graphs the two measures become closer.

We report the results attained by an experimental study designed to understand whether I_{max} is a reasonable bound according to two criteria: (a) the solutions attained should be of high quality, possibly reaching a limiting behaviour for the algorithm, (b) the computation time should not be excessive.

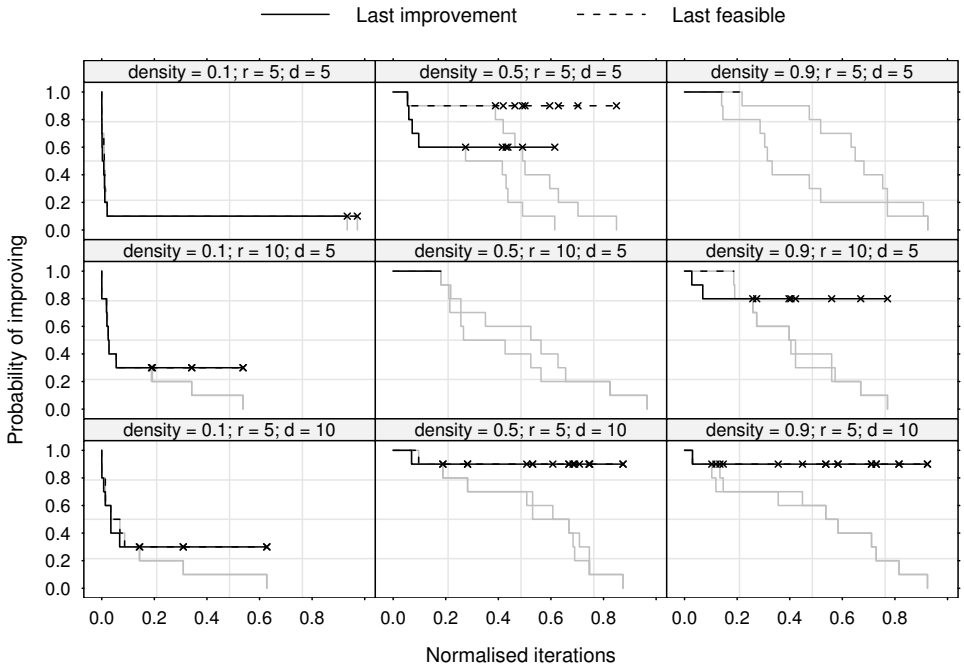


Figure 5.11.: The probability for GOF-TS of finding an improvement in random graphs of size 60. The distributions are obtained from 1 run on 10 graphs per instance class. The grey curves are the distributions at $10 \times I_{max}$ and the black ones are the censored distributions at I_{max}.

We first report indications relative to criterion (a). To this end, we run GOF-TS for $10 \times I_{max}$ iterations on the new random graphs described in Section 5.4. Due to the cost of solving some of these graphs, we restrict ourselves to consider graphs of size 60 and the following combinations of r and d: 5,5; 5,10; 10,5 (see Section 5.4 for the definition of these two parameters). Including the probability parameter p, the classes of instances considered are 9 and comprise 10 graphs each. The classes are recognisable in Figure 5.11. The grey curves represent the probability of a last improvement to occur at the indicated number of iterations, normalised by $10 \times I_{max}$. The black curves represent instead the same data distribution as if they were censored at I_{max} iterations. The grey distributions are meant to be only indicative, because their data would also be probably censored if we could run an experiment with $100 \times I_{max}$. Clearly, the entity of improvements missed at I_{max} iterations indicates that this bound is too low and that at least $10 \times I_{max}$ should be used.

In Figure 5.12 we observe that the choice I_{max} versus $10 \times I_{max}$ has an impact on the relative order of performance of the algorithms in our study. In other terms, the increase of computation time consequent to using $10 \times I_{max}$ rather then I_{max} is not equally profitable for all algorithms as the presence of crossing curves indicates.

On the other side we must control, according to criterion (b), that $10 \times I_{max}$ does not yield an excessively long computation time, both for us, for running the experiments, and for a possible planning scenario. To this end we check how the computation time of GOF-TS scales with instance characteristics.

In Figure 5.13 we report the empirical cumulative distribution functions of the computation time needed by GOF-TS to perform I_{max} and $10 \times I_{max}$ iterations on the 9 classes of instances. We observe that (i) the increase in computation time due to the multiplication

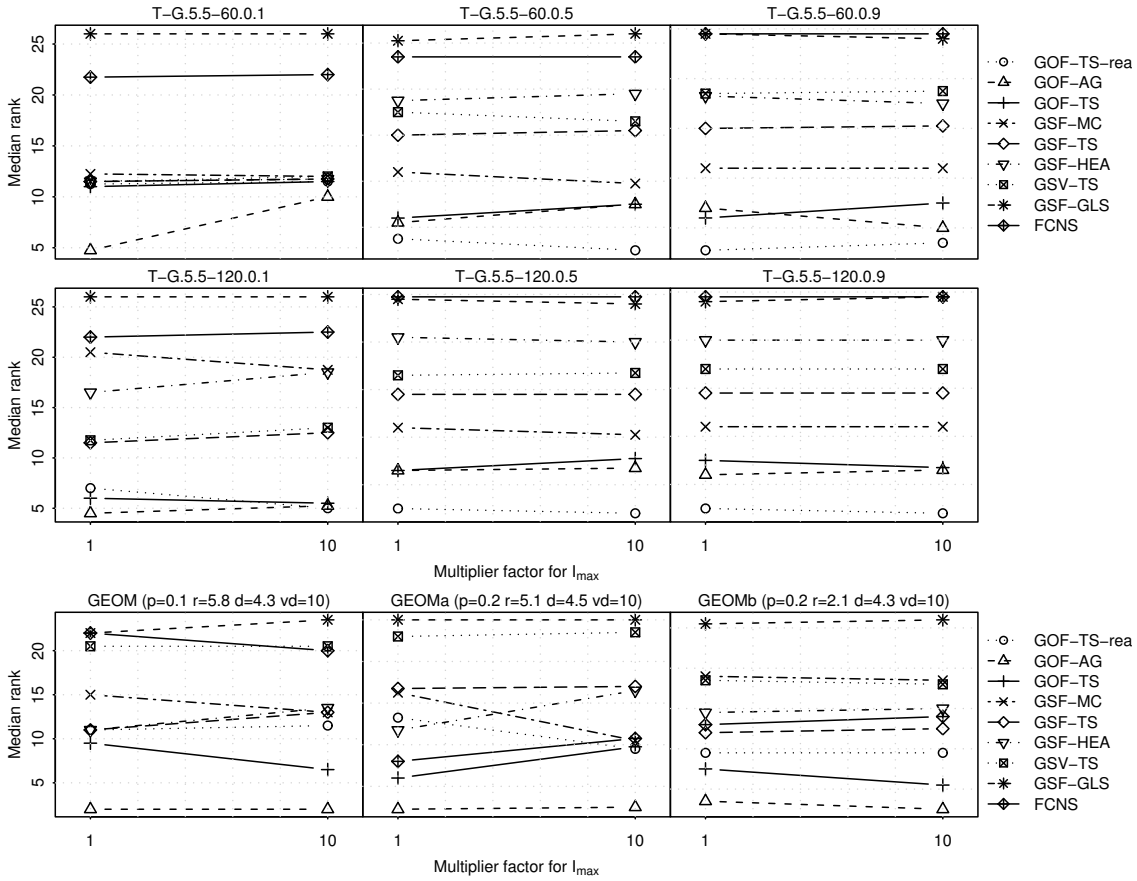


Figure 5.12.: The effect of increasing the running time on the algorithms. On the Uniform graphs, instances have sizes 60 and 120, and $r = d = 5$. On the Geometric graphs the title reports average statistics with p being the edge density, r requirements, d edge distances, and vd vertex distances.

of I_{max} is linear and follows the same multiplying factor, namely 10, (ii) the computation times for GOF-TS to solve the set T-colouring instances are much higher than those experienced for graphs of the same size in the GCP, (iv) the increase of the computation time due to the increase of the density is considerable, (iv) the number of colours required at each vertex has a stronger influence on the computation time than the separation distances, (v) on graphs from the lower right class a termination criterion of $10 \times I_{max}$ entails waiting 2 hours and 45 minutes for instances of only 60 vertices. In addition, we comment that GSF-TS and GOF-TS behave similarly, and hence from the computational point of view there is not significant difference in solving the problem in the split or the original graph. For more detailed numerical results we refer to the Tables in Appendix C.2.1.

In order to study the computational complexity of GOF-TS and obtain a model for predicting its running time on the basis of instance features, we try to derive from the collected data curve bounds which are a function of the 5 variables $|V|, \rho, \bar{r}, \bar{t}_{vv}, \bar{t}_{uv}$. McGeoch et al. (2002) discuss how to adapt known linear regression techniques to the study of the asymptotic behaviour of algorithms. By using two of the techniques there indicated by these authors, namely log-log and Box-cox, we obtain the models reported in Table 5.3. On the Geometric graphs the Box-cox technique did not returned an answer because of lack of convergence. On the Uniform graphs instead it returned a model which is exponential, although it indicated that the curve has to be regarded as an upper bound in the asymptotic behaviour. However an exponential model is not very useful. Considerations

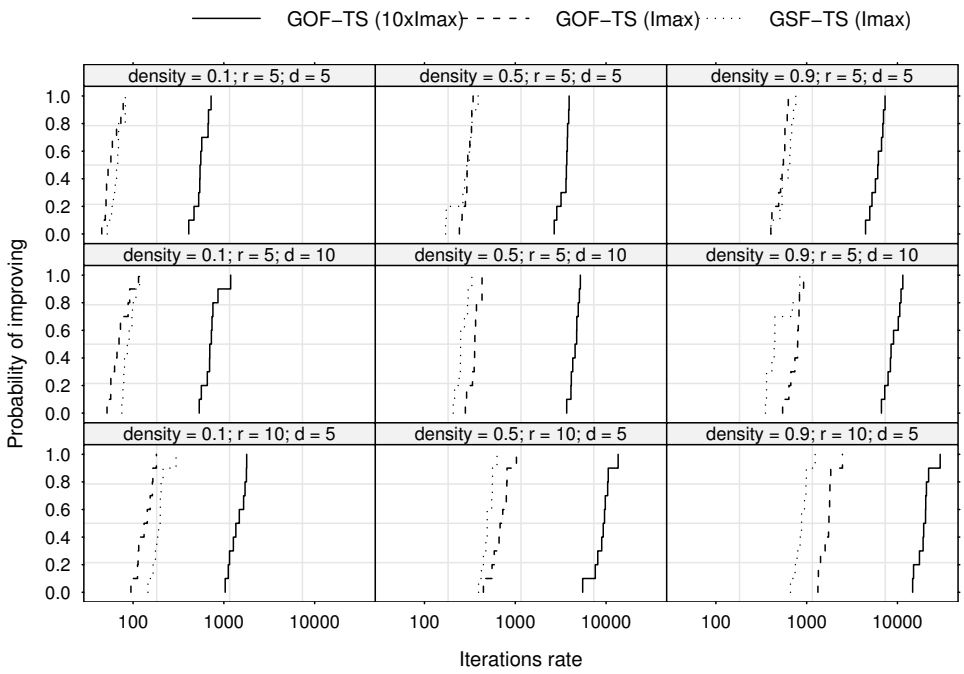


Figure 5.13.: Computation time for GOF-TS and GSF-TS for performing I_{max} and $10 \times I_{max}$ iterations on different graphs of size 60. The empirical cumulative distributions functions are obtained from 1 run on 10 graphs per instance class.

	Uniform	Geometric
log-log	$\lvert V\rvert^{2.36}\rho^{0.99}\bar{r}^{3.68}(\bar{t}_{uu}+\bar{t}_{uv})^{2.54}$ **l**	$\lvert V\rvert^{1.30}\rho^{0.12}\bar{r}^{1.30}(\bar{t}_{uu}+\bar{t}_{uv})^{8.23}$ **c**
Box-cox	$1.02^{\lvert V\rvert}13.06^{\rho}2.20^{\bar{r}}1.70^{\bar{t}_{uv}}$ **u**	
Selected	$t=\frac{1}{3.91\cdot 10^5}\cdot(\lvert V\rvert^{2.36}\rho^{0.99}\bar{r}^{3.68}(\bar{t}_{uu}+\bar{t}_{uv})^{2.54})-79.56$	$t=\frac{1}{5.12\cdot 10^{11}}\cdot 1.02^{\lvert V\rvert}13.06^{\rho}2.20^{\bar{r}}1.70^{\bar{t}_{uv}}$

Table 5.3.: The outcome of the linear regression analysis for determining the asymptotic time behaviour of GOF-TS in relation with instance feature. Conform to the notation introduced by McGeoch et al. (2002) we indicate with a letter l, c and u the type of bound. The letter l is used for lower bound, u for upper bound and c if the curve fits the data without a prevalent diverging trend.

on the complexity of GOF-TS (see Section 5.6.5) lead us to conjecture that an iteration of GOF-TS should have complexity $\mathcal{O}(|V|^a\rho^b(\bar{r}+\bar{T}_{vv}+\bar{T}_{uv}))$ and this looks more similar to the model provided by the log-log transformation. In the light of this fact, we decide to rely on the polynomial model provided by log-log. Its characterisation together with the coefficients is given in the last row of the table. Note that, in the case of random graphs, the model is a lower bound, hence, with an increase of computation times its predictions are actually under-estimates.

We will use this model, that predicts the computation time for performing I_{max} iterations, for determining the time limit on each instance of the experiments described next. For numerical indications about the predicted times and a comparison with the real ones we refer to the Appendix C.2.

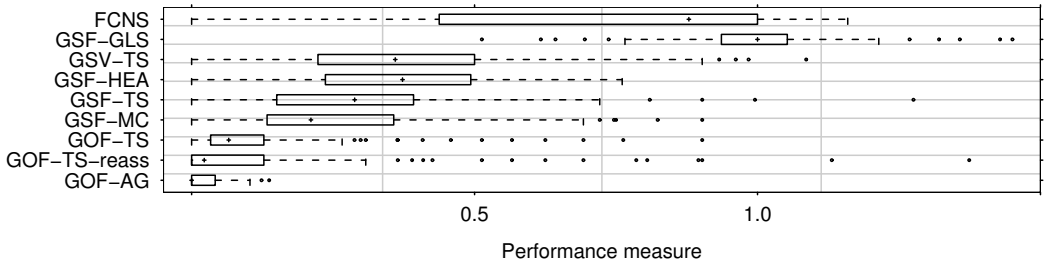


Figure 5.14.: Box-plot of performance error measure err on the 90 random graphs of size 60.

The design of experiments. We separate the two blocks of experiments relative to the two classes of instances, Uniform and Geometric random graphs. In the light of the results in the previous paragraph, we decided to use the time limit corresponding to $10\times$ I_{max}, hence, 10 times the values returned by the model of Table 5.3. To cope with the long computation time issue both the experimental designs *several runs on various instances* and *one run on various instances* were used. For the 90 Uniform random graphs in total we perform 1 single run per instance while for the 27 Geometric random graphs we perform 3 runs per instance. Note that running all the experiments for the 9 algorithms on our reference machine (Section 4.4) require 45 days (without taking into account preliminary experiments for the tuning).

Performance measurement. The SLS algorithms introduced are designed for minimising the span of the colouring. The algorithm response we measure is $k = \max\{\Gamma\}$, *i.e.*, the maximal number of colours used in the solution. Clearly, it is $\widetilde{sp}(G) = k - 1$. In order to normalise response among instances, we transform k into the error measure err defined by Equation 4.2 on page 129. Here, we substitute $\widetilde{k}^{ROS}$ with the median result returned by G-DSATUR and $\chi(G)$ with $\widehat{sp}(G)$. In Figure 5.14, we see that in some circumstances some algorithms are not able to improve over the results provided by the G-DSATUR heuristic. In particular GSF-GLS appears really not worthwhile to be used (obviously it cannot do worse than G-DSATUR, and values larger than one are due to the fact that the values of the heuristic are averaged). For graphs of large density, FCNS does not better than G-DSATUR. Besides this observation, the analysis through box-plots does not present any further evident indications and speculations about differences in the algorithm performance are not appropriate from these plots.

The choice of statistical tests. We checked the assumptions for the application of the statistical tests defined in Section 3.6.2. In all cases, diagnostic plots for checking the assumptions of parametric tests suggest that a parametric analysis is not appropriate. We rely therefore on permutation and rank-based tests. In Figure 5.15 and 5.16 we present the comparison of the two methods for determining confidence intervals in the all-pairwise comparisons of our algorithms. Figure 5.15 corresponds to a *"several runs on various instances"* while Figure 5.16 to the *"one single run on various instances"* scenario. In both cases, rank-based tests appear more powerful, although, in the two figures results are aggregated without considering the presence of stratification variables which may be used for reducing the variance. However, given the results of the analysis reported in Appendix B, and assuming that we give more importance to finding a better span in many

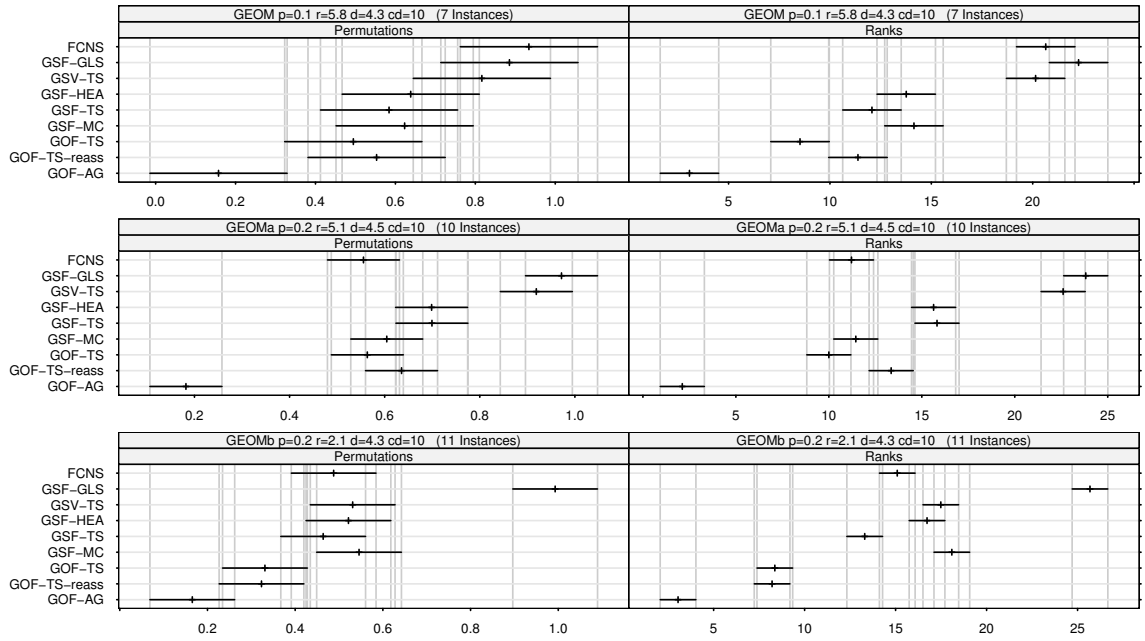


Figure 5.15.: Confidence intervals for the all pairwise comparisons of approximate algorithms on Geometric instances of the set T-colouring problem. Permutation tests with USP procedure are used on the left column while Friedman's rank-based tests are used on the right column. The experimental design is *"several runs on various instances"* with 3 runs per instance. The x-axis reports the average error measure and the average rank, respectively. The time limit applied corresponds to $10 \times I_{max}$ iterations of GOF-TS.

instances rather than large improvements only on few, we decided to base the comments that follow on the inference produced by rank-based tests.

The best approach to the problem. In Figure 5.15, algorithms are presented in increasing order of performance as the scale approximate the origin of the axes. It clearly arises that solving the problem in the split graph is not the best approach. All algorithms that worked well for the GCP are weak in performance when adapted to solve the set T-colouring problem with this representation. In particular GSF-GLS and GSF-HEA perform always significantly worse than GSF-TS. Surprisingly, among the algorithms that work on the split graph the best is GSF-MC which is never dominated by the other algorithms working on split graph and it performs significantly better as the requirements at vertices increase (which corresponds to an increase of vertices and clique sizes in the split graph). The use of k variable also does not appear a good choice to solve the problem, although our algorithm in this context was a quite simplistic one and perhaps better algorithms could be devised.

In contrast, solving the original graph, that is, maintaining the vertex-distance constraints distinguished from edge-distance constraints and imposing their satisfaction in the solution representation (thus restricting the neighbourhood to neighbours that do not violate such constraints) results to be the best solution approach. Indeed, 3 out of the 4 best methods, GOF-TS, GOF-TS-reass, and GOF-AG use this representation.

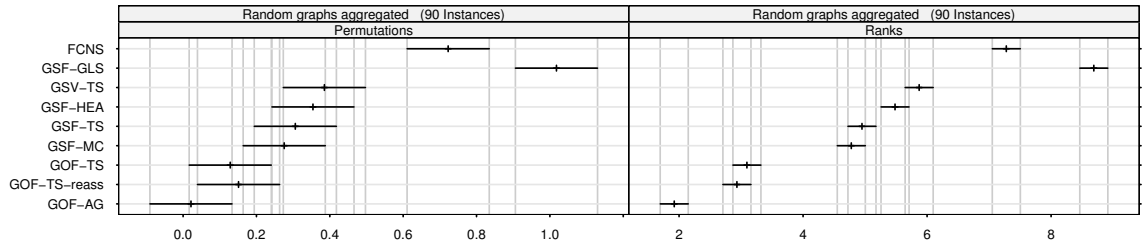


Figure 5.16.: Confidence intervals for the all pairwise comparisons of SLS algorithms on aggregated Uniform random instances of the set T-colouring problem (see also Figure 5.17). Permutation tests are used on the left column while the Friedman's rank-based tests are used on the right column. The experimental design is *"one single run on various instances"*. The x-axis reports the average error measure and the average rank, respectively. The time limit applied corresponds to $10 \times I_{max}$ iterations of GOF-TS.

The best algorithm and the influence of instance features. For Geometric graphs, the best algorithm is GOF-AG. This holds for all the three classes (Figure 5.15) and, consequently, also in the aggregate case. On the Uniform random graphs the aggregate analysis of Figure 5.16 indicates the same conclusion. However, on these graphs a detailed analysis reported in Figure 5.17, where columns represent different edge density and rows different requirements and distances constraints, shows that GOF-AG is not always the best on all classes. In the cases with edge density 0.5 or 0.9 and requirements per vertices below 5, GOF-TS-reass performs better. Important is the comparisons between these two algorithms and GOF-TS, since both of them might be seen as enhancements of this very simple algorithm. The results indicate that the use of an iterated greedy procedure or of the reassignment neighbourhood $\mathcal{N}_A$ are in general useful, although there are few cases where differences are not significant.

As far as FCNS is concerned, the number of iterations it performed in our experimental setting ranged from $2,4 \cdot 10^6$ to $5.4 \cdot 10^7$. In the original paper by Prestwich (2003) the bound was $3 \cdot 10^7$, hence, the results here discussed are in line with those presented by that author. Nevertheless, the results are rather weak and the method appears not to be competitive.

A detailed summary of the numerical results is reported in Appendix C.2. We just mention here that the results attained on the Geometric graphs are better than the best known results in the literature on 25 instances out of 28, reaching up to 36 colours less on `GEOM100a`, while they are worse only on 2 instances.

An analysis on GOF-TS-reass We try to understand the influence of instance characteristics on the behaviour of GOF-TS-reass. In Table 5.4, we report a collection of statistics that help us in this task. The number of improvements made possible by the function reassign of Algorithm 5.6 is considerable when compared to the number of improvements due to one-exchanges (we recall that Algorithm 5.6 is called only when no improving one exchange is found, hence the improvements would have been missed without the use of the restricted one-exchange neighbourhood $\mathcal{N}_1^R$). Particularly relevant for the feasibility of the algorithm, which has an exponential worst case, is the fact that the size of $|F|$ remains constantly small and comparable to the requirements. However, this is not an unlikely situation when solving instances close to their optimal span. The range of colours $f_{max} - f_{min}$ increases, instead, with the density and this fact could explain the

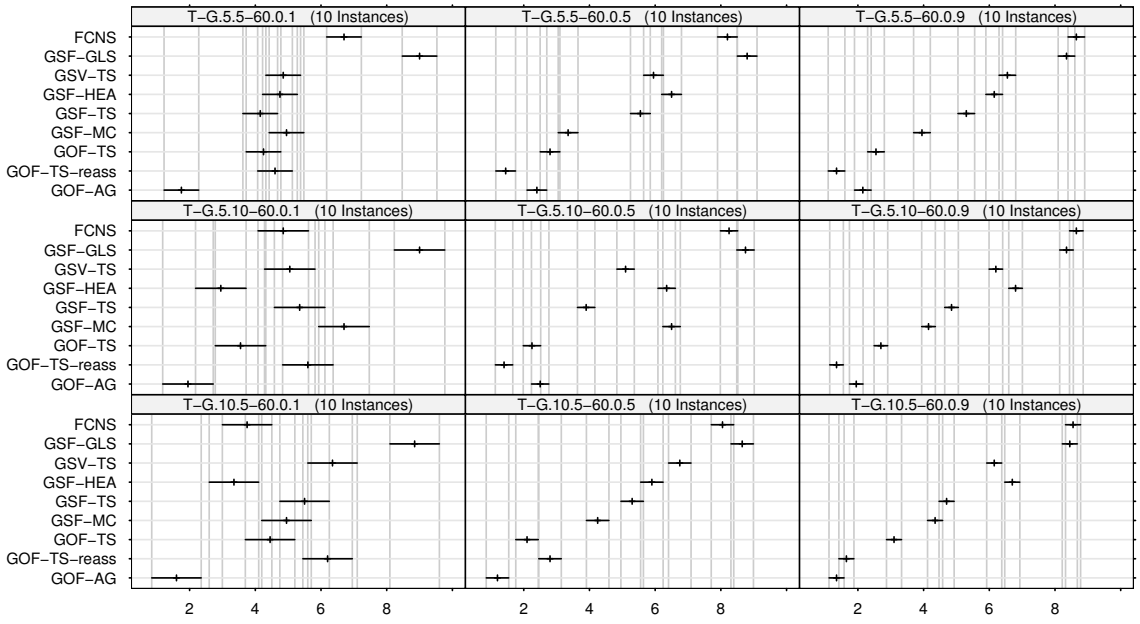


Figure 5.17.: All-pairwise comparisons on stratified Uniform random graphs. *Edge density* varies on the columns and *requirements and distances* on the rows.

better performances of GOF-TS-reass on graphs of larger density. With a larger range, it is indeed easier to find rapidly a feasible reassignment or, given that $|F|$ is small, to determine that no feasible reassignment exists. On the class T-G.10.5-60.0.1, we have a strange behaviour: the algorithm required much longer run times, as a consequence of the large $|F|$, and no improvement in the intra-vertex reassignment neighbourhood $\mathcal{N}_A$ was found. Yet, the improvements were few also in the restricted one-exchange neighbourhood $\mathcal{N}_1^R$ and hence this class exhibits some particular properties that would probably be interesting for further studies. In conclusion, since worse performance in comparisons with GOF-TS are only due to longer run times, a possible improvement for GOF-TS-reass would then be to limit the calls to reassign to cases where F is close in size to $r(v)$.

Scaling of performance with instance size. We designed another experiment considering Uniform random graphs of size 60 and 120 and $\bar{r} = \bar{t} = 5$ and we collected 3 runs per instance with a time limit corresponding to $10 \times I_{max}$, where the computation time for I_{max} is given by the model of Table 5.3. The all-pairwise comparison is reported in Figure 5.18. We observe that the significance of the differences the increases due to the passage from 1 to 3 runs per instance in the design of the experiment, and becomes significant even for edge density 0.1. Differences increase also with respect to the size of the graph. The relevant observation is that the good performance of GOF-TS-reass on graphs of edge density 0.5 is maintained with the increase of graph size. As we noted before, the complexity of Algorithm 5.6 depends on the entity of requirements at the vertices and the number of colours in F, but not by the size of the graph. The outcome is that GOF-TS-reass is the best algorithm for instances of edge density around 0.5 and requirements below 5. In an aggregate analysis (not reported) of the data represented in Figure 5.18

class	iter.	calls	improv. in $\mathcal{N}_1^R$	improv. in $\mathcal{N}_A$	$\lvert F\rvert$	$f_{max} - f_{min}$
T-G.5.5-60.0.1	8880	17698	1179	59	7	25
T-G.5.10-60.0.1	9259	15953	839	98	8	50
T-G.10.5-60.0.1	9994	20110	6	0	26	73
T-G.5.5-60.0.5	9030	25463	776	194	6	43
T-G.5.10-60.0.5	9187	22112	926	113	6	133
T-G.10.5-60.0.5	9287	35321	851	138	9	69
T-G.5.5-60.0.9	9232	21370	651	117	5	53
T-G.5.10-60.0.9	9807	13782	223	30	8	159
T-G.10.5-60.0.9	9437	23402	640	77	10	209

Table 5.4.: Statistics on GOF-TS-reass. The first four columns report statistics gained from 10000 iterations of GOF-TS-reass and are the median values out of 10 runs on one single instance per class. They represent the number of iterations in which there was no improving one-exchange and the function reassign of Algorithm 5.6 was called (iter.), the total number of calls to the function reassign (calls), and the number of improving moves in the two neighbourhoods. The last two columns report statistics obtained by one single run of 10000 iterations of GOF-TS-reass on a single instance per class. They represent the number of colours in F for the vertex v in reassign, *i.e.*, the colours which are feasible according to the edge-distance constraints of vertices adjacent to v, and the range of colours in F, *i.e.*, $f_{max} - f_{min}$, with $f_{max} = \max\{c : c \in F\}$ and $f_{min} = \min\{c : c \in F\}$.

there is no difference between GOF-AG and GOF-TS-reass, while they both perform better than GOF-TS.

We conclude noting that there is no indication against the conjecture that these results should hold also for larger instances with the same characteristics, as those solved by Dorne and Hao (1998b).

The importance of tuning. The tuning of the parameters is crucial with SLS algorithms. In Figure 5.18, we show its importance for GSF-MC. An aggregate analysis indicates in a tabu length of 20 the best choice but for instances of density 0.9 a length of 10 yields clearly better results. Similar considerations hold for GOF-TS where a δ value of 10 yields definitely better results than 20 for instances with high density (this should be true also for GOF-TS-reass although we did not extensively test it). Our original choice was determined by an aggregated evaluation of performance which was biased by the larger number of graphs of small edge density because of the presence of the Geometric graphs. The behaviour of GSF-MC has to be regarded as a confirmation that the tuning of SLS algorithms is a determinant factor for the success of a solution method and that it can have very important effects, not only marginal ones. A correct tuning requires the thorough understanding of the instance classes to which the algorithm must be applied and a good guess of a reasonable range of values for the parameters. A methodology that can help in this tuning process is the one based on sequential testing and racing approaches. This methodology is used extensively in Chapter 6. It is beyond the scope of our analysis here to fine tune the various algorithms for the set T-colouring problem and we leave this for future work.

The improvement over G-DSATUR. In the Appendix C.2, Table 5.5, we present the numerical comparison of the range of values for the lower bound, the G-DSATUR heuristic, and the best SLS algorithm on the Uniform random graphs. The ranges are determined by two stochastic factors: the graphs (10 per class) and the algorithm. Since we do not know how good the lower bounds are, we cannot comment on the hardness of these instances.

The improvements over the G-DSATUR heuristic are always relevant and clearly indi-

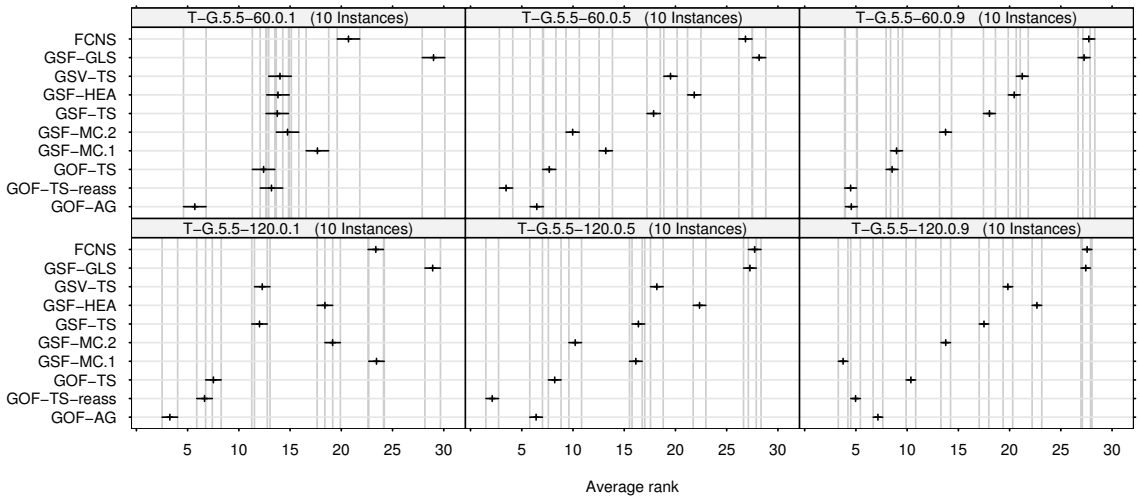


Figure 5.18.: All-pairwise comparisons on stratified random graphs. *Edge density* varies on the columns and *size* and distances on the rows.

Instance	Lower Bound (sec.)	G-DSATUR	GOF-TS-reass/GOF-AG
T-G.5.5-60.0.1	29 – 40 (2)	41 – 62	34 – 42
T-G.5.5-120.0.1	20 –53 (2)	58 –79	45 –57
T-G.5.10-60.0.1	42 – 68 (2)	78 – 107	63 – 83
T-G.10.5-60.0.1	55 – 91 (7)	79 – 116	62 – 93
T-G.5.5-60.0.5	30 – 50 (7)	89 – 116	74 – 89
T-G.5.5-120.0.5	41 –70 (9)	154–189	125–141
T-G.5.10-60.0.5	61 – 105 (8)	168– 231	137– 169
T-G.10.5-60.0.5	66 – 100 (22)	148– 222	123– 163
T-G.5.5-60.0.9	71 – 96 (41)	160– 194	137– 164
T-G.5.5-120.0.9	105–126(37)	276–321	247–276
T-G.5.10-60.0.9	91 – 125 (48)	304– 370	261– 291
T-G.10.5-60.0.9	141– 173 (117)	288– 353	253– 298

Table 5.5.: Comparison between lower bounds, construction heuristic, and best SLS algorithm (either GOF-TS-reass or GOF-AG) on the random graphs instances. The values refer to k, the number of colours used in the solution.

cate the importance of applying SLS algorithms. Even in the instances with $\bar{r} = \bar{t} = 5$ and density 0.1 where the distance from the lower bound is small, there is an improvement of about 10 colours. On these graphs, however, we noted that it is hard to distinguish differences among the algorithms (see Figure 5.17) and given the proximity of the best SLS algorithm results to the lower bounds we may conjecture that a "floor" effect is present. On all other classes, instead, a large gap lower bound–best solutions exists.

We conclude that the new instances introduced are sufficiently challenging and the improvements over G-DSATUR are important. Numerical details of the results on all these instances as well as on the Geometric ones are given in Tables C.8 and C.9 of Appendix C.

The minimal order problem. In Figure 5.19 we analyse the solutions returned by the algorithms under the criterion of minimising the order. When doing so, GOF-TS-reass results to rank among the worst algorithms, while GOF-AG and GOF-TS remain the best but are overtaken on instances with high edge density by GSV-TS. This latter result supports

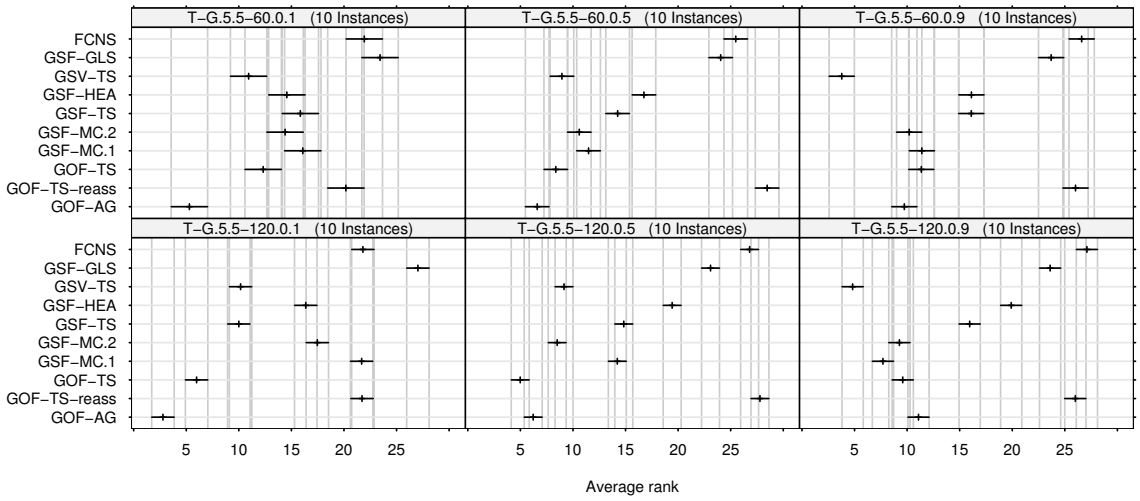


Figure 5.19.: All-pairwise comparisons on the *order* of the colouring for stratified random graphs. *Edge density* varies on the columns and *size* and distances on the rows.

the conjecture that minimising the order and minimising the span are two objectives in conflict with each other. The interaction between these two objectives may deserve further research. We add that in the development of GSV-TS, it has been quite problematic to derive a good objective function for the minimal span. In particular, the objective function $f = -\sum_{i \in \Gamma} |C_i|^2 + |V_s| \cdot |V^c|$ where V_s is the set of vertices in the split graph and V^c the vertices involved in at least one distance conflict, seemed to lead the search very rapidly to good solutions in terms of the order number.

5.7. Discussion

We defined graph colouring generalisations and studied SLS algorithms for the set T-colouring problem. We focused on two type of random graphs, *Geometric* and *Uniform*. The former are part of the DIMACS repository, the latter have been re-generated because those available were too large to allow a large scale experimental analysis on SLS algorithms. The new graphs generated amounted to 120 and varied in size, edge density, vertex requirements, and distance constraints.

The study on construction heuristics showed that a generalised form of DSATUR that solves the T-colouring transformation of the set T-colouring problem is the best choice on both classes of graphs. In terms of solution quality it performs better than the ROS and RLF heuristics adapted for solving the T-colouring problem and it is not dominated by a set of heuristics specific for the set T-colouring problem. In terms of computation times it remains very fast and solves split graphs of 3000 vertices in about 2.5 seconds.

The comparison of SLS algorithms comprised state-of-the-art algorithms and adaptations of algorithms from the GCP. The methods that work on the original formulation of the set T-colouring problem and solve a sequence of problems with fixed k resulted, with few exceptions, preferable to those working on the transformation of the problem in set T-colouring problem.

Clear indications arose that GOF-AG, an Adaptive Iterated Greedy based on Tabu Search with one-exchange neighbourhood, is the best overall algorithm both on the *Uniform* and on the *Geometric random graphs*. Our re-implementation of this algorithm yields much better performance than those originally presented by Lim et al. (2003), and we were able to improve the best known results on 25 out of 28 *Geometric random graphs*. However, a more fine-grained analysis of the instance classes revealed that for graphs with edge density 0.5 and requirements at the vertices below 5 colours, GOF-TS-reass, a Tabu Search algorithm with one-exchange neighbourhood enhanced by occasional exact reassignment of colours at the vertices, performs better than GOF-AG. These two algorithms can be easily combined together, and thus a further boost in performance is possible in future implementations.

As far as graph colouring generalisations are as concerned, our study contradicts some of the claims encountered in graph colouring literature, like the one that algorithms performing well for the GCP should perform also well for its generalisations. We showed that this is not true and that adaptations of methods like HEA and GLS resulted in algorithms that perform poorly on the set T-colouring problem.

The main difference with respect to the similar analysis on the GCP is the strong increase of computation times required by SLS algorithms for solving the set T-colouring problem. The choice for the time limit has been different in comparison with the previous chapter. We tried to infer a model from preliminary observations by means of linear regression techniques. The advantage of this approach might be the prediction of reasonable time limits for solving an instance from its features that are known *a priori*. On the one hand, this task resulted as being complicated, rich of pitfalls, and, to a certain extent, unsatisfactory in its results, mainly due to the large variance in the measures of computation times. On the other hand, the problem of deciding an appropriate stopping time for SLS algorithms still remains a "problem" (of course, when no practical limitations occur) and methods for the analysis of results that are less dependent from one particular such choice would be very useful in this context.

The main concern with the choice of the time limit is that it may affect the results. To this end, we provided evidence that some differences in the relative order may arise although these differences seem not to affect the best algorithms. The facts that we brought in support of this claim are two: (i) the two best algorithms are both based on GOF-TS, which exhibits a continuous improvement of solution quality over the run times that we could afford to test, (ii) the similarity of results at I_{max} and $10 \times I_{max}$ algorithm iterations.

Two other facts are remarkable to be pointed out. They confirm known pitfalls in algorithm testing. The first is that aggregating results may lead to rough conclusions. This became evident on the *Uniform random graphs* where a detailed analysis of the results revealed that, depending on the characteristics of the instances, different algorithms may result preferable. The analysis of algorithms should therefore take into account homogeneous classes of instances or instances that are well representative of the practical application for which the algorithm is designed. The second fact is that tuning is not a marginal factor but a determinant one in the performance of SLS algorithms. This suggests that in a real application, improved performance may be obtained by further tuning of parameters. The statistical methodology that we adopt in the next chapter indicates how the tuning process can be done effectively by reducing unnecessary testing. We observed, indeed, that some algorithms are significantly inferior already after few runs (*e.g.*, GSF-GLS) and they could be removed very soon to leave space for a better selection of the most promising ones or for the enlargement of the set of parameter values considered.

Our analysis is a compromise between the need of finding algorithms that behave well enough in general and the need of understanding how much performances may vary with respect to the classes of instances. An specific tuning of parameters on the large spectrum of instance classes considered would have been very costly, hence, we gave preference to robust tuning of the algorithms and attempted, where possible, to link the parameters to the characteristics of the instances thus making them part of the heuristic rules tested with the algorithm.

Finally, there are two main lessons learned in this chapter which are relevant for the next chapter where we will solve a complex timetabling problem. First, it is profitable to use a solution representation that reduces the search space and satisfies by construction some constraints. This is made evident by the fact that algorithms that solved the set T-colouring problem working on the split graph resulted clearly inferior in performance to all approaches that used the original graph and imposed the satisfaction of the vertex constraints. This reduction of the neighbourhood is reasonable since it maintains the search space connected. Secondly, exact algorithms to solve small assignment problems are appealing because they can find new improvements where basic neighbourhoods fail. This result contradicts somehow the outcome of the previous chapter where a very large scale neighbourhood search algorithm gave an opposite indication. It is wrong, therefore, to generalise this result and its validity must be verified in every specific case.

Chapter 6.

Course Timetabling

*In which we advocate an engineering approach for the application of SLS methods to real life problems and suggest the use of a systematic development methodology. We use timetabling as a case study to bring evidence that this approach is successful.**

6.1. Introduction

Timetabling can be generally defined as the activity of assigning, subject to constraints, a number of events to a limited number of time periods and locations such that desirable objectives are met as much as possible. Areas where such an activity is required are, among others, educational timetabling, employee timetabling, sports timetabling, transport timetabling, and communication timetabling.

Educational timetabling can be subdivided into three main classes: school timetabling, course timetabling and exam timetabling. Here we will be concerned with a particular case of university course timetabling. In university course timetabling a set of lectures must be scheduled into rooms and timeslots subject to constraints that are usually divided into two categories: "hard" and "soft". Hard constraints must be strictly satisfied with no violation allowed, while in the case of soft constraints it is desirable, but not essential for feasible timetables, to minimise their violations. Constraints and their importance differ significantly among countries and also among institutions (Carter and Laporte, 1998).

Typical in course timetabling, are the availability of a limited number of timeslots and the requirements of allocating lectures only into suitable rooms, having no more than a lecture per room. In some countries, like in Italy or in the United Kingdom where courses are composed of well defined lectures with different subjects and teachers, a further requirement is that lectures belonging to the same course and, hence, attended by the same students, are scheduled in different timeslots. Note that this latter constraint is rarely present in Germany, (at least not at the Computer Science Department of Darmstadt University of Technology), although this may not be a choice by rather a necessity. In many universities, timetabling is still done by hand and the involvement of any further constraints increases the complexity of this activity. An automatic system removes such limitations and leaves wider organisational freedom. It makes also possible to take into

*Part of the research described in this chapter was carried out together with Dr. Mauro Birattari, Dr. Olivia Rossi Doria and Krzysztof Socha. The content of the chapter has also been object of a joint article accepted for publication at the Journal of Scheduling.

account soft constraints such as the desirability of having lectures well spread through the week so that the students do not have to attend too many or too few courses on a day.

Course timetabling can be formulated as a combinatorial optimisation problem. Given that a large number of events (lectures) are to be scheduled and a wide variety of constraints is present, the problem to be solved can be very difficult and a variety of approaches from the field of Operation Research and Artificial Intelligence have been proposed. Many publications on applications of different techniques to timetabling appeared in recent years and the proceedings of a biannual International Conference on the Practice and Automated Timetabling (PATAT) collect the state-of-the-art. However, all timetabling systems are specifically designed for particular versions of a timetabling problem and comparisons between them are rarely possible.

Our scope, in this chapter, is broader than merely solving a specific timetabling problem. Timetabling is a real world application that exhibits issues common to many other real world problems and our intention is to show how we can tackle such an example effectively with SLS methods. The choice of timetabling as a representative case for real applications is due to (i) its high complexity and degree of specificity, which makes minimal the chance of finding reusable algorithms (this feature is typical of real life problems); (ii) its connection to graph colouring and its generalisations which gives us the basis to model and solve effectively timetabling problems; (iii) the possibility of measuring the effectiveness of our general approach in the International Timetabling Competition,[1] that was held in 2002 and 2003 and aimed at attracting the best researchers of the field. The selected problem has three hard constraints and three soft constraints, each representative of different types of constraints.

Michalewicz and Fogel (2000) argue in favour of the following approach to problem-solving:

> Better solutions to real-world problems can often be obtained by usefully hybridising different approaches. Effective problem-solving requires more than a knowledge of algorithms; it requires a devotion to determining the best combinations of approaches that addresses the purpose to be achieved within the available time.

We totally agree with this vision and follow it for solving our specific timetabling problem. Accordingly, in the case of SLS methods, we do not pledge ourselves in advance to any rigid method, such as one single metaheuristic, but make use of a combination of different methods. Yet, this poses additional problems in the configuration of the final SLS algorithm and in the selection of its constituent components. We suggest that proceeding in a systematic way with the strong use of experimental testing is the correct manner to develop such effective, hybrid algorithms, or, said in other words, to assemble an effective SLS algorithm from SLS components.

The novelty of our approach is the interactive use of a systematic experimental methodology based on sequential testing for the development of SLS algorithms. Sequential testing has recently been introduced to the tuning of metaheuristic parameters by Birattari et al. (2002) and we described it in Chapter 3. It is indeed a very useful tool for the whole

[1] The International Timetabling Competition was organised by B. Paechter and L. Gambardella from the Metaheuristics Network, and was sponsored by PATAT, the International Series of Conferences on the Practice and Theory of Automated Timetabling. As members of the Metaheuristics Network, we could submit an algorithm but we were not allowed to win the prize. For details on goals and rules of the competition we refer to the official web site `http://www.idsia.ch/Files/ttcomp2002/` (October 2004).

development process of SLS algorithms as it enables us to evaluate many candidate algorithms and it can be used in an automatic manner saving a lot of tedious hand-work. Moreover, if used sapiently, ongoing results can suggest the design of further algorithm candidates, which may be inserted in the testing procedure at any time. In this way, the sequential testing may be regarded as an interactive procedure and it may guide the development of effective algorithms that are highly specialised for the situation at hand. A further advantage of such a methodology is that it provides with some statistical guarantee that the final algorithm is a good algorithm. In a real context, where benchmark results are not available, this is indeed a contribution not to underestimate.

We used this methodology to develop an algorithm for submission to the International Timetabling Competition. The competition defined the instance class, the rules and the deadline for the entry of the submission. This situation is very similar to a real context, where these specifications are present and are part of the problem formulation. The competition had 24 feasible submissions from all over the world. According to the evaluation criterion of the competition, our algorithm outperformed the official winner which, in turn, outperformed by a quite large margin all the other submissions. We use this fact as evidence that our approach is profitable.

The chapter is organised as follow. In Section 6.2, we discuss the current state of art in educational timetabling and place our work within the context of that area. We introduce the specific timetabling problem that we solve in Section 6.3, and, in Section 6.4 we discuss possible formalisations of this problem as a graph colouring problem with the involvement of different types of constraints. In Section 6.5, we define the methodology for the SLS engineering process and describe its application in our practical case. Next, in Section 6.6, we provide the details of the final SLS algorithm, whose general framework is worth to be reused in similar applications, and analyse the impact of its components in Section 6.7. We conclude with a discussion on the guidelines for the application of SLS methods in Section 6.8 and with a summary of the main issues arising in this chapter in Section 6.9.

6.2. Methods for timetabling: the state of the art

Probably due to its academic environment, educational timetabling has received particular attention and it is the most fervid area for new ideas which are then applied also to other fields of timetabling such as staff scheduling and rostering (see Ernst et al., 2004 for a review on these related application areas).

Current research on university course timetabling focuses on three main directions. First, given the large variety of problems arising from the specific needs of different institutions, there is the attempt of formulating standard timetabling problems, which include supersets of constraints, where portable programs can be developed and compared. The web repository on Timetabling Problems maintained by L. Merlot[2] is an example of efforts towards this goal. Secondly, there is the tendency to develop generalised frameworks that can handle a wide range of problems or that provide the practitioners with as many components already implemented as possible. Examples are standardised models for handling constraints proposed by Chand (2004) and Custers et al. (2004) or libraries for metaheuristics such as EasyLocal++ (Di Gaspero, 2002), or other commercial packages

[2] L. Merlot. "Operations Research Group: Timetabling Problem Database." October 2003. http://www.or.ms.unimelb.edu.au/timetabling/. (June 2005.)

such as iOpt (Voudouris et al., 2001), SpaceMap (Burke et al., 2004a) and Harmony (Post and Veltman, 2004). However, as discussed in Section 2.4.5 and as also pointed out in this specific context by de Werra (1985), the idea of developing a universal timetabling program which could be used everywhere does not seem to be reasonable. The huge variety of timetabling problems makes it very unlikely that an effective heuristic method exists which is good for all constraints of different nature. Accordingly, frameworks leave to the final user the task of assembling and customising the solution methods. This process necessarily requires a certain degree of expertise and devotion of time from the user side. Therefore, there is currently a third direction of research, that looks at systems with many pre-developed algorithms and seeks for methodologies for choosing automatically and intelligently the appropriate algorithm for the problem at hand. The arrival point should be a system able to recognise the similarity of the new problem with previously solved problems and apply to the new problem the best technique for the matched case. Examples in this direction are *hyper-heuristic* approaches to select among heuristics proposed by Burke, Kendall, and Soubeiga (2003) and *case-based reasoning* systems recently introduced by Burke, Eckersley, McCollum, Petrovic, and Qu (2004b). The general trend in timetabling goes therefore towards the realisation of general systems which embed both solution techniques and automatic methods for their selection.

As far as solution techniques are concerned, the tendency seems to indicate SLS methods to be the most appropriate. Mathematical programming approaches like column generation are usually limited to a small number of constraints and small instances. A recent example of integer programming applied to a highly-constrained timetabling problem is given by Daskalaki et al. (2004). The main effort with these techniques lies in the modelling of the constraints and in the definition of the variables, because existing software packages can be used for solving the resulting integer programming formulation. Network flow methods are in general limited to simple assignments without constraints (see for instance de Werra, 1985). Otherwise, in some sporadic cases, the problem can be decomposed into a series of assignment type problems and the network flow model may become useful. With a similar idea, cluster methods split the events in subgroups, solve the problem for the subgroups, and then re-combine events together (Balakrishnan et al., 1992).

Applications of these classical techniques from Operations Research have, however, scarce impact in practice. Better performing alternatives are constraint programming techniques, that formulate the timetabling problem as a set of variables, each with its own domain, and as a set of constraints expressed in a well defined language (we refer to Zervoudakis and Stamatopoulos, 2001 for a contextualisation of constraint programming in timetabling). This approach is particularly appealing for solving the hard constraints but less for solving the soft constraints. Some successful applications, therefore, use constraint programming together with SLS methods which are, in contrast, very good at minimising soft constrains (see White and Zhang, 1998 and Merlot et al., 2003). The interest for SLS methods is very high, as can be seen from the large number of publications in every edition of PATAT which use these techniques. The main advantage of SLS methods is their extreme flexibility which makes them relatively easy to develop and quite good in performance in many contexts. Another advantage is that they can be used also in an online setting when the problem changes. This may be important in some timetabling problems. As, for instance, in employee timetabling where one employer can suddenly ask for vacation and the schedule must be updated in order to cover its duties. In this case, it is desirable to repair the schedule with the minimal necessary changes without having to reschedule the whole crew of employers and it can be easily done with local

search algorithms.

An issue in timetabling which receives increasing attention is how to combine the search for solutions and the decision-making process. Soft constraints are indeed a criterion to discern among solutions and their importance may not always be clear. This makes timetabling a multi-objective problem. The way in which these criteria are used depends on the specific situation. If the decision maker knows *a priori* a preference order between criteria, then solutions satisfying this order have to be found. In the opposite case, solutions are first found which exhibit a trade off between the various criteria and then the decision maker selects the most adequate. We can distinguish the following three alternative methods to evaluate solutions.

Combine objectives: this is the "classical" method widely used in highly constrained problems. It consists in the combination of the criteria into a single value. The use of weights and transformations allows to control the degree of compensation and the importance of each criterion. The majority of publications on educational timetabling fall into this typology. A variant to this method is suggested by Burke et al. (2001) who propose a similar method in which the value is obtained by measuring the distance of the objective value from an ideal point.

Alternating objectives: in this method a criterion at a time is optimised while imposing constraints on the others. It can be used when a priority order is defined. Several applications to timetabling have been proposed which are usually referred to as lexicographic or multi-phased approaches (see Thompson and Dowsland, 1998 and Paquete and Stützle, 2002a)

Pareto-based: in this method, the preference between solutions is established by the concept of dominance in the Pareto sense, that is, a solution is better than another if it is not worse in any criterion and there is at least one criterion in which it is better. Applications of this method to timetabling are rare and so far limited to two criteria. The main references are Carrasco and Pato (2001) and Paquete and Fonseca (2001) (who solved the problem at the same university).

For a deeper insight in all the issues related with multi-objective optimisation in timetabling we refer the reader to Silva et al. (2004). In our case study, the choice of the method is imposed by the competition and corresponds to a combination of objectives in a single evaluation function.

For a large collection of general timetabling applications, we refer to the proceedings of the five PATAT conferences organised in the last decade (Burke and Ross, 1996, Burke and Carter, 1998, Burke and Erben, 2001, Burke and De Causmaecker, 2003, and Burke and Trick, 2004 and to the recent special issue of the European Journal of Operational Research[3]. Relevant to our work is the high specificity of the constraints in timetabling which motivates the choice of flexible methods such as SLS methods. Our approach to the problem is based on the conviction that timetabling requires necessarily a certain degree of algorithm customisation and that a theory of practice for this task is needed.

[3]European Journal of Operational Research, Volume: 153, Issue: 1, February 16, 2004

6.3. The definition of the problem

The university course timetabling problem (UCTP) that we consider was proposed in the context of the International Timetabling Competition and was also treated in the research of the Metaheuristics Network. It is an abstraction of a real UCTP arising at Napier University in Edinburgh and we denote it as UCTP-C. It was proposed by Ben Paechter who also made available a set of randomly generated instances.

Problem description

The UCTP-C can be formalised as follows.

> ***Input:*** *A set of events*[4] E*, a set of* 45 *timeslots* T *(*5 *days of* 9 *periods each), a set of rooms* R *in which events can take place, and a set of students* S*. Every event has associated a list of students* L^E *who have to attend it. In addition, each event has a set of required features* F^E*, and each room has a size* S^R *and a set of satisfied features* F^R*.*
>
> ***Question:*** *Which* timetable *schedules all events in timeslots and in rooms in such a way that the following three hard constraints* $H = \{\mathsf{H1}, \mathsf{H2}, \mathsf{H3}\}$ *are satisfied:*
>
> H1*: only one event is assigned to each room at any timeslot.*
>
> H2*: the room is big enough for hosting all attending students and satisfies all the features required by the event;*
>
> H3*: no student attends more than one event at the same time;*
>
> *and the sum of the violations of the following three soft constraints* $\Sigma = \{\mathsf{S1}, \mathsf{S2}, \mathsf{S3}\}$ *are minimised:*
>
> S1*: a student should not have a class in the last slot of a day;*
>
> S2*: a student should not have more than two classes in a row;*
>
> S3*: a student should not have a single class on a day?*

In the UCTP-C, an assignment is *infeasible* if it violates one of the hard constraints. An infeasible assignment is considered worthless and it is therefore discarded. To help in the presentation we define $\mathcal{A}$ the set of all possible complete assignments, and $\tilde{\mathcal{A}} \subseteq \mathcal{A}$ the set of feasible assignments that satisfy the constraints in H. A feasible assignment should minimise the sum of the violations of constraints in Σ. To this end, an *evaluation function* $f(a)$ is defined that associates to each assignment $a \in \tilde{\mathcal{A}}$ a penalty cost $f(a)$ that accounts for violations of constraints in Σ. The goal is to find an assignment $a^* \in \tilde{\mathcal{A}}$ of minimal cost, that is, an assignment such that $f(a^*) \leq f(a)$ for all $a \in \tilde{\mathcal{A}}$.

The evaluation function $f(a)$ is defined by the International Timetabling Competition. The quality of a feasible assignment $a \in \tilde{\mathcal{A}}$ is given by:

$$f(a) = \sum_{s \in S} \big(f_1(a, s) + f_2(a, s) + f_3(a, s)\big)$$

where S is the set of students and

[4] Without loss of generality, we call *event* the single indivisible activity which constitutes the object of scheduling. Alternative names used in the terminology of course timetabling are *class* or *lecture* of a course.

$f_1(a, s)$ is the number of times a student s under assignment a has a class in the last timeslot of the day.

$f_2(a, s)$ is the number of times a student s under assignment a has more than two consecutive classes, each time weighted by the number of classes exceeding two in a sequence (for example, $f_2(a, s) = 1$ if s has 3 consecutive classes, $f_2(a, s) = 2$ if s has 4 consecutive classes, etc.)

$f_3(a, s)$ is the number of times a student s under assignment a has to attend a single event on a day (for example, $f_1(a, s) = 2$ if s has two days with only one class).

The smaller $f(a)$ is the better is the assignment judged in quality.

The benchmark instances

The instances of the competition were generated by a random generator and reflect real-world UCTP instances at Napier University in Edinburgh. The details on how the generator works were kept secret in order not to unveil shortcuts to the solution. The only indication available was that at least one assignment with an evaluation function value equal to zero exists for each instance.[5] In Table 6.1, we report the main statistics of the 20 benchmark instances which were used to evaluate the submitted algorithms. The number of timeslots is fixed to 45 while events range between 350 and 440 and students between 200 and 300. The number of rooms suitable for each event (rooms/event) is quite low. Recalling the experience with vertex distance constraints in the set T-colouring problem, this fact may suggest that an exact algorithm could be feasible for the assignment of rooms to scheduled events. The number of events per student (events/student) and the number of students per event (students/event) are, instead, informative of how difficult might be satisfying all constraints. Since it is known that for each instance a zero cost assignment exists, for constraint S3 there exists a feasible assignment that uses only 40 timeslots. Indicatively, with 400 events and 10 rooms, this corresponds to filling up completely the 40 timeslots available with events.

The evaluation of candidate algorithms in the competition

In the International Timetabling Competition, the participants were asked to submit an algorithm and a seed number for re-producing the solution deemed the best on each of the 20 instances. The winner was the participant whose solutions scored the best results across all instances. More precisely, each participant j was associated a penalty value p^j computed as:

$$p^j = \sum_{i \in I} \frac{f_i(a_i^j) - f_i(a_i^b)}{f_i(a_i^w) - f_i(a_i^b)}$$

where I is the set of the 20 instances of the competition, f_i is the evaluation function value for instance i, a_i^j the assignment submitted by participant j for the instance i, and a_i^b and a_i^w are, respectively, the best and the worst assignment for the instance i among

[5] After the competition we learned that the generator works in a backward form, starting from an optimal solution and defining an instance therefrom.

Instance identifier	1	2	3	4	5	6	7	8	9	10
# events	400	400	400	400	350	350	350	400	440	400
# students	200	200	200	300	300	300	350	250	220	200
# rooms	10	10	10	10	10	10	10	10	11	10
rooms/event	1.96	1.92	3.42	2.45	1.78	3.59	2.87	2.93	2.58	3.49
AV(events/student)	17.75	17.23	17.70	17.43	17.78	17.77	17.48	17.58	17.36	17.78
SD(events/student)	1.07	1.16	1.13	0.97	1.16	1.10	1.07	1.04	0.97	1.04
AV(students/event)	8.88	8.62	8.85	13.07	15.24	15.23	17.48	10.99	8.68	8.89
SD(students/event)	1.82	3.16	1.93	5.26	5.97	5.59	11.36	5.88	3.98	1.86

Instance identifier	11	12	13	14	15	16	17	18	19	20
# events	400	400	400	350	350	440	350	400	400	350
# students	220	200	250	350	300	220	300	200	300	300
# rooms	10	10	10	10	10	11	10	10	10	10
rooms/event	2.06	1.96	2.43	3.08	2.19	3.17	1.11	1.75	3.94	3.43
AV(events/student)	17.41	17.57	17.69	17.42	17.58	17.75	17.67	17.56	17.71	17.49
SD(events/student)	1.14	0.93	1.07	1.13	1.08	1.02	1.01	0.97	1.02	1.05
AV(students/event)	9.58	8.79	11.05	17.42	15.07	8.88	15.15	8.78	13.28	14.99
SD(students/event)	2.87	5.26	5.37	9.20	6.58	1.93	5.99	2.20	4.22	3.59

Table 6.1.: Descriptive statistics for the instances of the competition; averages and standard deviations are denoted, respectively, by AV and SD.

all those submitted. In other terms, for each instance a participant was given a penalty value between 0 and 1; 0 if his assignment resulted to be the best for that instance, and 1 if the assignment was the worst. The total penalty value was then obtained by the sum of the single values on the instances. The winner was the participant that scored the lowest final penalty value.

The maximal time available to solve an instance was defined by a benchmark code which had to be run on the same machine as the algorithm. The times produced by the benchmark code are reasonable times for desktop computers, for example, on a Pentium III with clock speed 700 MHz the time available was about 18 minutes.

6.4. Timetabling and graph colouring formalism

In his "Introduction to Timetabling", de Werra (1985) models school timetabling as an edge colouring problem[6] and course timetabling as a vertex colouring problem. In the latter case, each event is associated to a vertex and edges are introduced between pairs of vertices if the corresponding events cannot be scheduled in the same period. A feasible course timetable in p periods will then correspond to a p-colouring in such a *constraint*

[6]Edge colouring can be considered as a special case of vertex colouring: edge colouring of a graph G is nothing else than the vertex colouring of the *line graph* of G. The line graph $L(G)$ of G contains one vertex v_c for each edge e of G, vertices v_e and v_f are connected if e and f have a common vertex. However, for solving an edge colouring problem it is almost always advantageous not to consider it as a general vertex colouring problem, but to make use of special properties of edge colourings.

graph, that is, every vertex receives some colour (period) and no two adjacent vertices (events) are allowed to have the same colour.

We can simplify our UCTP-C into this representation by ignoring for the moment the assignment of rooms.

Definition 6.1 *The* constraint graph $G_T(E, D)$ *relative to the UCTP-C is constructed by associating an edge to every vertex pair* e_i *and* e_j *if they correspond to vertices that share one or more students.*

Clearly, finding a feasible colouring for the constraint graph G_T, in the general case, is an $\mathcal{NP}$-complete problem (Garey and Johnson, 1979). However, this is not enough to create a feasible solution for the UCTP-C, since the constraints H1 and H2 are not considered. Definition 6.1 can be further extended.

Definition 6.2 *The* constraint graph $G'_T(E, D')$ *relative to the UCTP-C is the graph where* D' *is the set of edges obtained by the union with* D *from Definition 6.1 and the set of edges between each vertex pair* e_i *and* e_j *that correspond to events that can be scheduled only in one single room which is equal for both.*

Studying the hardness of solving a graph colouring problem in the constraint graph gives an idea of how hard might be finding a feasible solution (one that satisfies all constraints H) for a UCTP-C instance. In Table 6.2, we report the statistics of the graphs G_T and G'_T derived from the UCTP-C benchmark instances. We also report the minimal number of colours. $\widehat{\chi}(G)$ and $\widehat{\chi}(G')$, found by the algorithm $\mathsf{TS}_{\mathcal{N}_1}$ when applied to those graphs. We observe that the differences in terms of colours by considering G_T or G'_T are relevant (in 13 instances this difference is more than 8). For the graph $G'_T(E, D')$, a feasible colouring with much less than 40 colours is always easily found, except on instances 3, 10, 12, and 17, where 40 colours must be used. Translated in terms of timetabling, the number of colours found per each instance represents the minimal number of timeslots required by a feasible assignment. The maximal number of timeslots used by an optimal solution is necessarily 40, hence the instances where 40 is also the lower bound appear as particularly difficult.

This graph colouring model is still quite distant from the requirements for solving the UCTP-C. The room assignment is, indeed, still not taken into consideration. The first implication when re-considering the presence of rooms is that the number of events assigned to timeslots cannot be higher than the number of available rooms. A graph colouring generalisation known as *bounded colouring* (Hansen et al., 1993) can be used for modelling this further constraint. In a bounded colouring, each colour class C_i has a bound on its size c_i and the number of vertices assigned to it must be lower than this value.

Once a feasible bounded colouring has been found, it would be possible to assess its feasibility with respect to the UCTP-C in a second step by matching the events scheduled in each timeslots with the available rooms. If the matching is not possible, then a new bounded colouring must be found. The procedure could iterate by trial and error until a feasible solution is found.

This was indeed the strategy adopted by the official winner of the competition Kostuch (2004). In his 3-phase approach, however, a bounded colouring was constructed by a heuristic and no successive local improvement was applied, thus, constantly, a set of vertices remained unassigned after the exact matching and a rather complex improvement process was needed to make the timetable complete and feasible.

Instance identifier	1	2	3	4	5	6	7	8	9	10
AV$(d(G_T))$	72.28	71.50	77.49	84.25	91.53	90.30	68.01	63.77	66.64	77.06
AV$(d(G'_T))$	81.00	83.00	93.67	90.28	108.51	90.95	72.10	67.83	75.01	80.25
SD$(d(G'_T))$	21.14	26.46	20.92	32.25	37.11	32.38	40.40	31.68	26.86	19.56
AV$(\bar{d}(G_T))$	0.18	0.18	0.19	0.21	0.26	0.26	0.19	0.16	0.15	0.19
AV$(\bar{d}(G'_T))$	0.20	0.21	0.23	0.23	0.31	0.26	0.21	0.17	0.17	0.20
SD$(\bar{d}(G'_T))$	0.05	0.07	0.05	0.08	0.11	0.09	0.12	0.08	0.06	0.05
$\widehat{\chi}(G_T)$	23	23	24	27	27	28	28	24	24	25
$\widehat{\chi}(G'_T)$	39	38	40	32	39	28	34	28	36	40

Instance identifier	11	12	13	14	15	16	17	18	19	20
AV$(d(G_T))$	73.39	69.81	72.01	82.11	78.10	71.13	86.25	69.33	80.11	84.53
AV$(d(G'_T))$	81.78	81.56	83.14	85.90	86.43	79.29	107.94	83.95	80.68	86.04
SD$(d(G'_T))$	24.08	44.96	33.19	47.80	38.70	19.57	32.95	23.91	30.06	23.77
AV$(\bar{d}(G_T))$	0.18	0.18	0.18	0.24	0.22	0.16	0.25	0.17	0.20	0.24
AV$(\bar{d}(G'_T))$	0.20	0.20	0.21	0.25	0.25	0.18	0.31	0.21	0.20	0.25
SD$(\bar{d}(G'_T))$	0.06	0.11	0.08	0.14	0.11	0.04	0.09	0.06	0.08	0.07
$\widehat{\chi}(G_T)$	24	25	25	28	26	25	27	25	25	25
$\widehat{\chi}(G'_T)$	34	40	35	29	35	36	40	39	25	28

Table 6.2.: The instances of the competition from a graph colouring perspective. Given are statistics on the vertex degree d and normalised vertex degree $\bar{d}$ of the two graphs G_T and G'_T and their chromatic number approximations attained by $\mathsf{TS}_{\mathcal{N}_1}$.

A further refinement is possible to try to include the room assignment and all hard constraints in a graph colouring formulation, such that a solution to that problem is a feasible solution for the UCTP-C. We end up with a variant of list T-colouring.

Let T be the set of timeslots and R be the set of rooms; timeslots and rooms are represented without loss of generality by integer numbers. Then we can represent the assignment of a timeslot and a room (t, r) by means of a unique value $v = t \cdot |R| + r$. The set $\Gamma = \{1, \ldots, |T| \cdot |R|\}$ corresponds then to the set of colours. We wish to find a mapping $\varphi : E \rightarrow \Gamma$ such that all hard constraints for the UCTP-C are satisfied.

To this end, we introduce a constraint graph $G''_T(E, D'')$ and further conditions that reflect the constraints of the UCTP-C and we require that φ be a feasible colouring under those conditions.

- Due to H1, the constraint graph $G''_T(E, D'')$ is completely connected and a collection of distances is associated with all edges: $\mathcal{T} = \{t_{ij} = 1,\ \forall(e_i, e_j) \in D''\}$. Accordingly, each event cannot receive exactly the same room in the same timeslot.

- Due to H2, a list $L_i \subseteq \Gamma$ of admissible colours is associated to each vertex $e_i \in E$. The list contains the value v that corresponds to rooms that are feasible for the event. For example, if $|R| = 10$, $|Q| = 45$, and $r = \{1, 3, 5\}$ are the rooms suitable for event e_i then it is $L_i = \{1, 3, 5, 11, 13, 15, \ldots, 441, 443, 445\}$.

- Due to H3, a second collection of distance constraints $\mathcal{T}' : \{t_{ij} = 1,\ \forall(e_i, e_j) \in D'\}$ is associated to the set of edges $D' \subseteq D''$ determined by the events that cannot be scheduled in the same timeslot (D' is the same as in graph G' introduced

in Definition 6.2). In order to satisfy this set of constraints, φ must be such that $||\lfloor\varphi(e_i)/|R|\rfloor - \lfloor\varphi(e_j)/|R|\rfloor| \geq t_{ij},\ \forall(i,j) \in D^{'}$.

However, when also soft constraints are to be considered, the models become even more complicated and the benefit of coding our UCTP-C as graph colouring for then solving it with the algorithms presented in the previous two chapters is lost. Nevertheless, the effort of modelling a problem into a simpler one which is well known and studied in the literature is not in vain. In contrast, this is the process that is at the basis of the design of SLS components. The identification of similarities with some known simpler problems permits the retrieval of a series of relevant information (*e.g.*, which construction heuristics, neighbourhood structures, etc. are appropriate) by consulting some opportunely organised collection of applications and results of SLS algorithms on standard problems. Unfortunately, such a collection does not yet exist, mainly due to the difficulty inherent in the organisation and representation of the knowledge of entities such as combinatorial problems. Examples that may inspire developments in this direction exist in the two close areas of computational complexity, with the book of Garey and Johnson (1979), or approximation theory, with the online compendium to the book of Ausiello et al. (1999).

Back to our case, many of the algorithmic components present in the algorithm that won the International Timetabling Competition and that we will describe later are extensions of components which where used in the graph colouring problems. The process that led us to model our UCTP-C as graph colouring problems consisted in the removal of constraints. Once the SLS components for the standard problem at the core have been identified the constraints must be re-introduced. This implies the adaptation of the SLS components. In this case, the further distinction of constraints in local and global constraints may be useful.

Local constraints are those whose violations can be checked quickly because they require only the single assignment to be known. SLS methods deal well with this kind of constraints by maintaining some static auxiliary data structures, as lists or matrices, and afterwards often they can check violations in linear time. In the UCTP-C, all the hard constraints and also constraint S1 of not having a student with a lecture in the last timeslot of a day are local constraints. In contrast, *global constraints*, do not manifest any precise bad individual assignment: anything appears acceptable, as long as all the local constraints in the solution are not violated. Violations of global constraints can be checked only once the solution or some part of it are complete and hence known. SLS methods deal less well with global constraints, which, for example, cannot be satisfied by solution representation or by neighbourhood restriction. Checking global constraints is typically computationally more expensive than checking local constraints. Auxiliary data structures are a good idea for speeding up this process but they are to be dynamic and their update is often complex to implement. Two soft constraints in the UCTP-C represent global constraints: not having students with two classes in a row, (S2) that can be checked while building a solution; and not having a student with only one class on a day (S3), that can only be checked when the timetable is complete and all events have been assigned a timeslot.

6.5. An engineering methodology for SLS methods

A central concept that we put forward in the previous chapters is the vision of SLS methods as a combination of components. In SLS algorithms for the GCP this combination was quite basic: a Construction Heuristic, an Iterative Improvement algorithm, and a Metaheuristic. The importance of maintaining algorithms relatively simple was determined by the fact that the GCP is one of the standard problems on which SLS components must be deeply analysed because they are highly reusable. Yet, we saw that some hybrid algorithms such as the hybrid Evolutionary Algorithm or the Iterated Local Search, both based on Tabu Search, are very competitive. Analogously, on the set T-colouring, the Adaptive Iterated Greedy which is often the best algorithm, is, in fact, a combination of two metaheuristics: Adaptive Greedy and Tabu Search. Even the algorithms $\mathsf{TS}_{\mathsf{VLSN}}$ on the graph colouring and GOF-TS-reass on the set T-colouring can be seen as been obtained by combination and hybridisation of more neighbourhoods.

This background leads us to support the view that a final, high-performing SLS algorithm in a context, in which re-usability is not a main concern, is obtained by the combination of more than three basic components. The final outcome is an hybridisation of parts that showed to behave well on standard problems, such as the GCP, TSP, MAX-SAT, etc. Assembling these parts is an engineering process consists in the adaptation of the components to the real life problem at hand, in their organisation and in their selection. A methodology is, clearly, needed to guide in this activity.

An important part of component-based engineering methodologies is the specification of the components. This task, we believe, has been quite well accomplished in recent years. In the previous chapter we distinguished components at different levels. Besides the distinction of Construction Heuristics, Iterative Improvements, and Metaheuristics at the highest level, we saw that within each of these constituents it is possible to recognise more fine-grained distinctions, as for example search strategy, neighbourhood structure, etc. The book of Hoos and Stützle (2004) is perhaps the best reference for a thorough description of fine-grained SLS components, their scope, and their implementation. Nevertheless, in spite of examples on the standard problems, the development and application of these components on a new problem progresses, in a certain extent, by intuition and by trial and error and has to rely on empirical testing, analysis of results and consequent actions. In software engineering, this process is known as *experimental development* and implies the refinement of algorithms step-by-step through several development cycles substantiated by empirical observations. Clearly, the definition of correct procedures for carrying out this process might be very helpful.

In this section, we propose a methodology and introduce appropriate tools which we validate through the application in the context of the International Timetabling Competition. The central point followed here is the adoption of the racing algorithm proposed by Birattari et al. (2002). We show that its use solves efficiently the problem of selecting the best among a wide spectrum of possible algorithmic configurations (the configuration problem). Moreover, it can also be used to acquire knowledge on the problem at hand and on the application of SLS methods.

The whole methodology is particularly suitable for the development of optimisers for solving real-world problems with expectations in terms of functionality, cost, and delivery schedule.

6.5.1. The methodology

We consider the application of SLS methods from a perspective of *algorithm engineering*. It comprises the design, analysis, implementation, tuning, debugging, and experimental evaluation of computer programs. Demetrescu et al. (2003) point out the need for defining standard methodologies and tools for the best accomplishment of these tasks. Similarly, in the context of SLS methods, algorithm engineering must proceed through the following steps.

Modelling and Requirement Analysis. It consists in modelling the real world system as an optimisation problem, thus identifying the problem, its constraints and its objectives. If there are multiple criteria for evaluating a solution it is crucial to understand whether there exist priorities. The precise specification, eventually in mathematical terms, of the problem is crucial, because it corresponds to the definition of the optimiser and the quality of its solutions. Moreover, it makes possible the communication within a team of theoreticians, developers, analysts, and managers to whom the problem has been commissioned.

To this phase also belongs the understanding of the requirements that the optimiser must satisfy. The application scenario (design, control, planning, as described in Section 3.3) determines the termination criterion and influences the design of the algorithm and its analysis. But also important (and so far not really understood) is the collection of training instances which are representative of the class of instances for which the optimiser is to be devised.

This phase is carried out in strict collaboration with the client who commissioned the optimiser. The advantage offered by the flexibility of SLS methods is that the definition of the optimiser can be "pulled" by the needs of the customer rather than "pushed" by the limitations of the solving techniques.

Brainstorming and Design. It consists in choosing the solution methods for the problem. This is the creative phase in which easily solvable sub-problems are recognised, similarities with standard problems are identified, but also new approaches are conceived. Standard problems may inspire successful construction heuristics, solution representation, neighbourhood structures, etc. Functioning algorithms are designed and metaheuristics added on top. The focus is on the general algorithmic structure without being too much involved into details, like data structures and coding. Usually, the outcome of this phase is a collection of ideas whose effectiveness is impossible to forecast at that point. We deem important in this phase to look at the problem outside of any technique framework, and to search for ingenious ways to solve it.

Coding. It consists in transforming the designs of the previous phase into prototypes. Profitable in this context are frameworks and libraries. The time invested to become familiar with these tools will pay off as soon as features will be added to the system. Even if the developers prefer to write their own code, it is profitable, for the organisation of the work, to use a framework. In this context, we encourage the use of EasyLocal++ whose organisation of objects was largely adopted by the author of this thesis. The chief advantage of this framework, rather than making available a library of methods for metaheuristics, is an appropriate definition of classes which allows to easily add components of any kind. Such an organisation becomes extremely important in complex algorithms, like the one for timetabling described in

this chapter, and in code maintenance tasks. Moreover, it makes feasible team work and easy the involvement of novices. A similar, alternative framework, is HotFrame (Fink and Voß, 2002). Both these frameworks allow the use of data structures like the Standard Template Library of C++ or the LEDA libraries.

Experimental Analysis. It comprises the tuning and comparison of algorithms. This is the phase in which statistical tests may be used. The theory of experimental design is helpful for the organisation and analysis of experiments. Full factorial or fractional factorial designs may be adopted when factors are clearly identified and orthogonal, *i.e.*, all combinations of their levels are feasible. This procedures entail a fine-grained analysis and are typically performed to find patterns that may then be exploited. They require therefore a certain degree of involvement and interaction. More likely, when the interest is only in determining which is the best algorithm, a single treatment factor design with the algorithms as treatment is the appropriate design. In this case the use of sequential testing, such as the racing algorithm is recommendable. It allows the comparison of many different alternatives in reasonable time and it does not require too much involvement because the process can easily be made automatic. Note that the adoption of a fully automatic race, in which all algorithms are instantiated at the beginning, allows to bypass that tedious part in algorithm development which in the literature is usually referred to as "preliminary experiments", consisting in the understanding of how the algorithm behaves with respect to computation time and tentative parameters. This phase may be very demanding because it imposes a strong interaction with the developer and very poor in its intellectual content. Furthermore, we believe that the racing approach can be very stimulating, balancing cooperation and competition within a work team composed by several persons. Other advantages brought by a racing methodology are the possibility of testing for bugs using large scale experiments, and the extraction of knowledge on the problem which other approaches like analytical analysis, landscape analysis, or qualified run time distributions are equally likely to provide.

Documentation and Maintenance. Documenting the internal design of the algorithm and maintaining and enhancing the optimiser are two further tasks required in practical contexts. The details of this task, however, are beyond the scope of this thesis.

The steps discussed above could be followed in a waterfall process. Yet, this is usually never the case because the results of the experiments suggest new, previously unseen directions. The realistic approach is therefore an iterative process consisting in the construction of an initially simple optimiser and in its evolution through further refinements and additions of components, once the promising ones have been determined. The racing approach is again very helpful. New candidate algorithms can be inserted in the race at an intermediate stage, run for a number of times equal to the runs collected for the other candidates in the race, and directly included in the tests.

Next we describe the use of this methodology in the context of the UCTP-C. We already described the *requirements* which are imposed by the competition and gave the formal *specifications* of the problem. The part relative to the aspects of coding is not relevant to our discussion. There exists a large body of literature on this issue which falls into the area of software engineering. Two publications in this area with focus on optimisation are Di Gaspero (2002) and Weihe (2001) which could then inspire further readings. We focus, instead, on the design and on the organisation of the experiments.

6.5.2. Design of SLS algorithms in the UCTP-C case

The UCTP-C has been part of the research activities of the Metaheuristics Network. Initially, simple construction heuristic and local search algorithms were developed by Rossi-Doria et al. (2003a). These construction heuristics and local searches served as a basis for comparing straightforward implementations of metaheuristics like Tabu Search, Simulated Annealing, Iterated Local Search, Evolutionary Algorithms, and Ant Colony Optimisation. All metaheuristics were bounded to use the same local search procedure.

The instances for the comparison were different from those of the International Timetabling Competition, although created by the same random generator, and were grouped into three classes according to their size. The results of the comparison were in part inconclusive. The performance of the metaheuristics varied between instances of different sizes and with respect to the capability of solving hard or soft constraints. The main pattern found was that Tabu Search and Iterated Local Search behaved better than other metaheuristics with respect to the capability of reaching feasibility but once a feasible assignment was found, Simulated Annealing appeared as the best choice for minimising the violations of the soft constraints. For an extensive description of these analyses and its results we refer to Rossi-Doria et al. (2003b) and to Chiarandini and Stützle (2002). It arose clearly, however, that restricting metaheuristics to the use of the same local search was not a correct approach for comparing them in practice because in this way not all their potential was exploited.

It was then decided to proceed in a different way and to develop a highly competitive algorithm to be submitted to the International Timetabling Competition.[7] The work was organised within a team of researchers of the Metaheuristics Network, each researcher being an expert on different metaheuristics. We first collaborated to exchange ideas and knowledge on the problem. Then it was decided to proceed in a competitive manner, and each researcher was left free to develop the technique of his expertise. The racing algorithm was used for assessing the algorithms thus giving a feedback to the developer on whether improvements were still feasible. In this way, the potential of each metaheuristic approach was exploited at best. In particular, the author of this thesis worked in detail on Simulated Annealing and Iterated Local Search. However, from the previous results it was clear that these methods had to be combined with others.

The following are the algorithmic components that were attentively considered within the team.

Construction heuristics. 60 different construction heuristics were implemented. They were inspired by graph colouring heuristics and other timetabling applications (see Carter et al., 1996, Burke et al., 1995, and Newall, 1999).

Local searches. Various local searches based on 7 different neighbourhood structures and different ways to explore the neighbourhood were analysed (besides those described in detail in Section 6.6.5 other such attempts are described in Rossi-Doria et al., 2003a). *Variable neighbourhood descent* was adopted for assembling sequences of local search procedures based on different neighbourhood structures.

[7] We put emphasis on the argument of the competition because it exemplifies the characteristics of a realistic context. In practice, indeed, there is a complex problem delivered by a client that has to be solved within a limited time horizon, there are few instances to run the experiments, and we are asked to deliver an algorithm with some guarantees on its performance, possibly the best that one can obtain. Such characteristics are typical in the three business companies the author has been involved: Cybertec, Eurobios, and AntOptima.

Metaheuristics. Preference was given to the following methods:

- *Tabu Search*;
- *Simulated Annealing*;
- *$\mathcal{MAX}$-$\mathcal{MIN}$ Ant System* a variant of *ant colony optimisation* (see Socha, 2003; Socha et al., 2003 for details on its application to the UCTP-C);
- *Heuristic Memetic Algorithm*, a variant of *Evolutionary Algorithms* that uses construction heuristics for the initialisation of the population as well as for the recombination of individuals and local search after every generation.
- *Iterated Local Search* based on variable neighbourhood descent, random moves perturbations, and a Simulated Annealing type acceptance criterion;
- *Iterated Greedy Search* is inspired by the homonymous method for graph colouring (Culberson, 1992); this algorithm tries to overcome local optima by destructing and reconstructing a part or the whole assignment.

It is immediately clear that testing exhaustively so many possible combinations of components, many requiring additional parameters to be set for finding the best possible configuration, is a task that goes far beyond the actual computational power available. An ingenious experimental methodology is therefore needed in order to maintain the correct approach of experimental testing and to avoid the pitfall of relying only on the developer's intuition.

6.5.3. The racing algorithm for the experimental analysis

The goal of the experimental analysis for our UCTP-C can precisely be stated: finding the best algorithmic configuration for the set of instances of the International Timetabling Competition having about one month of time available for running the experiments. The class of instances is well defined but the number of instances available is small. The experimental design most suitable is therefore the *"several runs on various instances"* design.

Deciding the number of runs for each pair (algorithm, instance) is a crucial decision which can be avoided by using sequential testing. The application of this methodology to our case corresponds to an extension of the racing algorithm called *F-RACE* introduced by (Birattari et al., 2002).

We recall that in sequential analysis an initial set of candidate algorithms is evaluated after a certain number of runs on all the available instances. Poor candidates are discarded if there is statistically sufficient evidence against them and the candidates left run again on the same instances. The elimination of inferior candidates speeds up the procedure (since less candidate algorithms need to be tested on the instances) and allows a deeper analysis of the most promising ones (see also Section 3.7). In the case under study, a *stage*, *i.e.*, the elementary experimental unit, consists in testing all surviving candidates on the available instances, with a single run on each instance. At the end of each stage a decision is made on which candidate configuration should be discarded.

In the competition, ten instances were made available since the beginning (Set A), while other ten were published only 15 days before the deadline for the submission of the algorithms (Set B). The first phase of the race consisted of one single *stage* and was conducted on the Set A of instances. Therefore, each of the candidates performed at least

10 runs, one on each of the 10 instances, before being discarded. When the Set B of instances was released, we were ready to start the second *stage* of our race with a number of surviving candidates that had already dropped to one sixth of the initial number. From then on, each stage of the race consisted in running once each surviving candidate on each of the 20 available instances.

From the second stage, the statistical tests used were the Friedman test of Equation 3.18. If the null hypothesis of no difference among all candidates was rejected, we proceeded with all-pairwise comparisons based on the extension of the Friedman test of Equation 3.19. In the first stage, the Friedman procedures of Equations 3.7 and 3.11 were substantiated by the Wilcoxon matched-paired signed rank tests with Holm's adjustment. After the 20th stage, two configurations were not significantly different. We based, then, our final selection on the indication of the exact binomial test on best results only, collected for the two algorithms on each instance during the whole race.

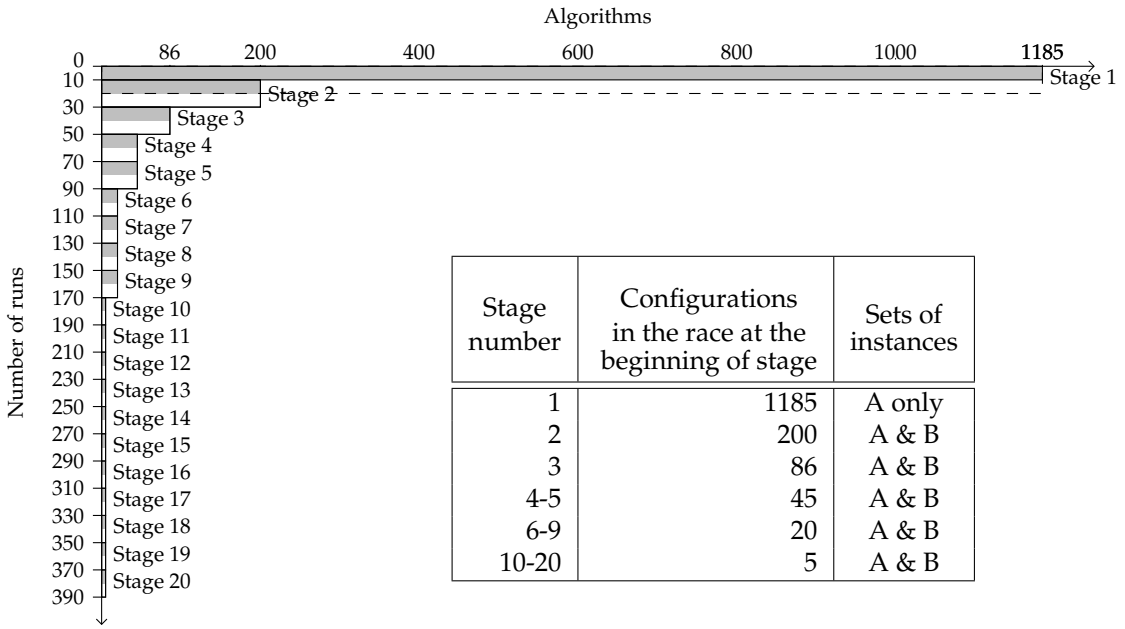
Contrary to the original *F-RACE*, which is a completely automatic procedure, we used in this context an *interactive* approach in which statistical tests and graphical visualisations were used at each stage for general indications arising from the results. On the basis of those indications it was possible, at the end of each stage, to insert new candidates combining features or refining good performing candidates already tested. Newly entered candidates were run on each instance for a number of times equal to the current number of runs before taking part in the test for survival. In our UCTP-C case we had an initial number of 879 candidate algorithms while the total number of candidates that the selection procedure considered reached a final value of 1185. This interactive approach constituted a helpful feature of our experimental testing. Indeed, the algorithm that finally won the competition was one of these newly inserted configurations.

Figure 6.1 reports graphically the details of the race in the UCTP-C case. The total surface covered by the gray and white bars represents the total amount of computation time required by the racing algorithm. The whole procedure took about one month of alternating phases of computation, analysis, and human decision, and involved up to a maximum of 30 PC running Linux, with CPU clock speed in the range between 300 and 1800 MHz, each using its own benchmark time given by the competition benchmark code (running the race on a single machine – like the one mentioned in Section 4.4 – would have required 270 days, more than 9 months). In the Figure, the surface of the race is equal to the one covered by the dashed box: this indicates that a brute-force selection method in the same time would have performed only one single run of each of the initial candidates on 18 instances. The race permitted, instead, to collect 390 runs for each of the 5 best configurations that reached the last stage, corresponding to 19 runs for each instance of the Set A and 20 runs for each instance of the Set B. Running all the 1185 configurations 19 times on all the 20 instances would have required about 15 years on a single machine.

At the end of the race, we had still time to run the selected candidate 77 more times on each of the 20 available instances (Set A + Set B) to choose the best solutions to submit to the competition out of 96 runs per instance.

6.5.4. General indications from the race

As already mentioned, the race can be used in a broader concept than merely determining the best algorithm. It can also be used indeed for gaining insights of more general significance with respect to the problem and algorithms. Den Besten (2004) uses this



Stage number	Configurations in the race at the beginning of stage	Sets of instances
1	1185	A only
2	200	A & B
3	86	A & B
4-5	45	A & B
6-9	20	A & B
10-20	5	A & B

Figure 6.1.: Graphical visualisation of the race including all configurations which were finally tested. The y-axis reports the number of runs collected for each individual configuration. Runs on the Set A of instances are represented in gray, while runs on the Set B are represented in white. Stage 1 was an exception in the race because it considered only instances of Set A. The dashed box is relative to a brute force approach which would take the same computational time as the whole race. The table provides in a numerical form the same information conveyed by the graph.

approach systematically in his study of iterated local search algorithms for scheduling problems. He suggests that candidates can be grouped by common features, such as, for example, candidates which use Simulated annealing and whose only difference is in the different initialisation of parameters, or candidates based on ant colony optimisation but with different population size. The survival rate during the race is then computed for each candidate and analysed. The groups, whose survival rate drops to zero after few steps of the race, use very likely a wrong approach to solve the problem.

The following observations arose in our case study from such a kind of analysis.

- The local search scheme we describe in detail later in Section 6.6.5 is preferable to the one proposed by Rossi-Doria et al. (2003a) and the acceptance of a certain number of non worsening moves in all local search procedures contributes to improve the final solution quality.

- The use of variable neighbourhood descent strongly enhances local search, as its presence was common in all candidates with high survival rate.

- Concerning the approach to solve the constraints, it was clearly preferable to keep hard and soft constraints separated, first finding a feasible solution and proceeding next to minimise soft constraint violations while maintaining feasibility. This approach was preferable over weighted sum combinations of hard and soft constraints violations.

- Algorithms which alternate local search with other heuristic mechanisms, like perturbations in iterated local search, reconstruction in iterated greedy, or recombination in evolutionary algorithm, work better if the feasibility is not broken during these passages. In general, reproducing feasibility was found to be very expensive in terms of soft constraints. (Since no way for doing this in iterated greedy was found, its survival rate remained very low.)

- Tabu Search for the soft constraints had also a low survival rate in spite of several attempts to reinsert it in the race with improved features. Its main drawback is the need of exploring exhaustively a neighbourhood which is too large. No effective way to restrict considerably the neighbourhood was found.

- Population based metaheuristics perform poorly when not enhanced with strong local searches. This is shown by the low survival rate of Evolutionary Algorithms and $\mathcal{MAX}$-$\mathcal{MIN}$ Ant System. The latter method, however, enhanced by a powerful local search exhibits the same high survival rate as the finally selected algorithm based on Simulated Annealing. This fact may be regarded as an indication that the choice trajectory-based or population-based is not determinant for the final success of the algorithm and that other choices are more important.

- Finally, some indication was given that the choice of an initial solution from a set of solutions constructed by several construction heuristics performs better than the same algorithm with one single initial solution. The difference between these two configurations arose only at the last stage.

Some candidates were also run only with the purpose of gathering specific pieces of information. For instance, candidates made solely of variable neighbourhood descent were used to understand which was the best order for the neighbourhoods. Single local searches were also used as candidates to distinguish between different examination strategies (*i.e.*, best improvement against first improvement or whether side walks moves should be accepted).

In general, such candidates confirmed that the use of problem specific knowledge is very important. Insights to restrict the neighbourhood of local search to only promising solutions, recognition of subproblems to which fast exact algorithms can be applied, ingenious ways to perturb or construct a solution, known theoretical results concerning the problem or subproblems were all elements which provided improvements over basic implementations. Finally, code optimisation played also a strong role and the fast evaluation of neighbour solutions by considering only local changes has a tremendous impact on the performance of all SLS algorithms. Stronger candidates with high survival rate were obtained by combinations of all these elements.

6.6. An effective algorithm for course timetabling

In the present section, we describe and analyse the final algorithm obtained by the use of the methodology described. We proceed in a top-down approach, first giving a high level view, then going through the details of the sub-procedures.

Function UCTP-C-Solver(I_k) %$I_k \in I$ is an instance of the competition
Preprocessing();
$\widehat{\mathcal{A}} \rightarrow \emptyset$;
for $i = 1$ to $|\mathcal{H}|$ %$\mathcal{H}$ is the set of construction heuristics
do
  $a_i \leftarrow$ Build_Assignment(h_i) %$h_i \in \mathcal{H}$
  $a_i \leftarrow$ Hard_Constraint_Solver(a_i);
  $a_i \leftarrow$ Look_Ahead_Iterative_Improvement(a_i);
  $\widehat{\mathcal{A}} \rightarrow \widehat{\mathcal{A}} \cup \{a_i\}$;
end
$a_{best} \leftarrow$ Select_Best_Assignment($\widehat{\mathcal{A}}$);
$a_{best} \leftarrow$ Soft_Constraint_Optimiser(a_{best});
return a_{best}

Algorithm 6.1: High-level pseudo-code for the UCTP-C Solver.

6.6.1. High level procedure

The algorithm starts by creating several feasible assignments, selecting the most promising one and improving it until the maximal computation time is exceeded. Algorithm 6.1 schematically reproduces this procedure that can be summarised in three steps: first a group of initial assignments is constructed by Build_Assignment; next each assignment is made feasible by the Hard_Constraint_Solver and assessed by a fast local search Look_Ahead_Iterative_Improvement. Finally, the best assignment is selected (Select_Best_Assignment) and further improved by means of procedure Soft_Constraint_Optimiser.

Build_Assignment constructs one assignment for each construction heuristic $h_i \in \mathcal{H}$, where $\mathcal{H}$ is a set of 60 different construction heuristics. Preliminary experiments showed that the initial assignment produced by such heuristics is never feasible, hence, the procedure Hard_Constraint_Solver is used to make them feasible. The two sub-procedures Hard_Constraint_Solver and Soft_Constraint_Optimiser are two local search based procedures that deal with hard and soft constraints, respectively. When trying to find a feasible solution, soft constraints are not taken into account, while when minimising the number of soft constraint violations, it is not allowed to break any hard constraint.

6.6.2. Data management

Some data provided by the instances can be processed at the start of the algorithm to obtain more synthetic information. In particular, the features and sizes for the rooms, S^R and F^R are useful solely to determine which rooms are suitable for each event on the basis of L^E and F^E. In addition to this, we use these data to create a matrix representation of the graph G'_T determined by the events that cannot be scheduled in the same timeslot according to Definitions 6.1 and 6.2. We also maintain some redundant data structures which are useful for speeding up the neighbourhood exploration in the local search. The overall data are:

- a matrix of size $|E| \times |R|$ where each entry indicates whether a room is feasible for an event.

Rooms \ Timeslots	T_1	T_2		T_i		T_j		T_{45}
R_1	-1	E_4	$\cdots$	$\mathbf{E_{10}}$	$\cdots$	E_{14}	$\cdots$	-1
R_2	E_1	E_5	$\cdots$	E_{11}	$\cdots$	$\mathbf{E_{15}}$	$\cdots$	-1
R_3	E_2	E_6	$\cdots$	$\mathbf{E_{12}}$	$\cdots$	-1	$\cdots$	-1
$\vdots$	$\vdots$		$\vdots$		$\vdots$		$\vdots$	
R_r	E_3	E_7	$\cdots$	E_{13}		E_{16}	$\cdots$	-1

Figure 6.2.: The assignment matrix representation. Events are assigned to rooms and timeslots; for example, event E_1 is assigned to room R_2 in timeslot T_1, while room R_1 has no event assigned in the same timeslot. Bold face is used for emphasising the events used in the example to describe Kempe chain interchange (see text).

- a list for each room with the events that can take place in it,
- a matrix of size $|E| \times |E|$ representing the adjacency matrix of the graph G'_T,
- a list for each event with the students that attend it,
- a matrix of size $|S| \times |E|$ where each entry indicates if a student has to attend an event.

The Standard Template Library of C++ is used for defining and handling these data structures.

6.6.3. The assignment representation

An assignment of events to rooms and timeslots can be represented by a rectangular matrix $X_{|R|\times|T|}$, such that rooms correspond to rows and timeslots to columns, and $|R|$ and $|T|$ are the number of rooms and timeslots, respectively. A number $k \in \{-1, 0, \ldots, |E| - 1\}$, corresponding to one of the $|E|$ events or to none of them (-1), is assigned to each cell of the table. If $X_{ij} = k$, then the event E_k is assigned to room R_i in timeslot T_j, if $X_{ij} = -1$, then the room R_i in timeslot T_j is free. With this representation, we assure that there will be no more than one event in each room at any timeslot, which implies that the constraint H1 will always be satisfied. An example assignment is given in Figure 6.2.

The room assignment can be done through the local search algorithm conjointly to the assignment of timeslots or it can be done by an exact algorithm once events have been assigned to a timeslot. In this latter case, the problem to be solved is a *bipartite matching* (Papadimitriou and Steiglitz, 1982). Bipartite matching can be treated as a max flow problem and it can be solved in $\mathcal{O}(\sqrt{|V|}|C|)$, where V is the union of the set of events assigned to the timeslot and of the set of possible rooms for the events, and C is the set of possible connections between the events and the rooms at hand (Sedgewick, 1988; Rossi-Doria et al., 2003a). Nevertheless, we preferred a breadth first search augmenting path algorithm, which, despite its higher worst case complexity of $\mathcal{O}(|V||C|)$, was considerably faster on the small matching instances that we have to solve in the UCTP-C (from Table 6.1, the number of rooms involved in the matching never exceeds 11 and the feature rooms/event is always a small value). The analysis of this algorithm as well as its

implementation is due to Setubal (1996). We identify it as Matching. We will make explicit when the matching algorithm is used, as, by default, we assume that the assignment of rooms is done conjointly to the assignment of timeslots in the local search procedure.

6.6.4. Construction heuristics

To build initial assignments we use construction heuristics, which were mainly developed in a separated work by Rossi-Doria and Paechter (2003). These are sequential algorithms that insert events into the assignment matrix one at a time. At each step, first a still unscheduled event is selected and then a pair timeslot-room for it is decided. The process can be presented as follows:

Step 1. Initialise the set $\hat{E}$ of all unscheduled events with $\hat{E} = E$.

Step 2. Choose an event $E_i \in \hat{E}$ according to a heuristic.

Step 3. Let $\hat{X}$ be the set of all positions for E_i in the assignment matrix in which the number of violations of constraints H2 and H3 is minimised.

Step 4. Let $\bar{X} \subseteq \hat{X}$ be the subset of positions of $\hat{X}$ in which the cost due to violations of the soft constraints Σ is minimised.

Step 5. Choose an assignment for E_i in $\bar{X}$ according to a heuristic.

Step 6. Remove E_i from $\hat{E}$, and go to step 2 until $\hat{E}$ is not empty.

Two heuristics, one to choose which event to insert next in the timetable and one to choose the position in the matrix for the chosen event, are needed at Step 2 and Step 5, respectively. We considered 6 heuristics to choose the next event and 10 heuristics to choose the position in the matrix, *i.e.*, the pair timeslot-room. Recalling that two events are adjacent if there is an edge between them in the graph G'_T, the different heuristics at Step 2 choose:

- an event with a maximal number of adjacent events;
- an event with a maximal number of adjacent events weighted by the number of students shared;
- an event with a maximal number of students and a maximal number of adjacent events;
- an event with a minimal number of possible rooms and a maximal number of adjacent events;
- an event with a maximal number of required features and a maximal number of adjacent events;
- an event with rooms suitable for a maximal number of other events and a maximal number of adjacent events.

Unresolved ties are broken by label order (an alternative choice would be to use random selection).

The set $\bar{X}$ is implemented as an ordered list of pairs room-timeslot which represent positions in the assignment matrix X where violations of the constraints H2, H3 and Σ are minimised; higher priority is given to H2 and H3. We note that $\bar{X}$ is not empty, otherwise the instance is unsolvable because it has more events than places available. The list is ordered by labels in case of unresolved ties. The following heuristics are used to select the pair room-timeslot from $\bar{X}$:

- the smallest possible room and ties broken in favour of a timeslot with the largest number of events already assigned to it;
- the least used room and ties broken in favour of the timeslot with the lowest label number;
- the latest timeslot in the last day and ties broken in favour of the room suitable for the least events;
- the room suitable for least events and ties broken in favour of the latest timeslot;
- the room suitable for least events and ties broken in favour of the timeslot with the lowest label number;
- the timeslot among those earliest in the day that has most parallel classes and ties broken in favour of the room with lowest label number.
- the timeslot according to a pre-established timeslot order that tries to keep difficult events spread out in the timetable and not in adjacent timeslots (we use three different orders, suggested by Terashima-Marín et al., 1999, that give rise to three different heuristics), and ties broken in favour of the room with the lowest label number;
- the pair with lowest label number in the list.

All 60 possible combinations of these heuristics are used in the algorithm. Preliminary experiments did not indicate any of them as being clearly dominating the others (see also the discussion in Section 6.7).

6.6.5. Iterative Improvement

Once a complete assignment has been created, it is made feasible by local search methods. Local search methods are also used for minimising the number of soft constraint violations when a feasible colouring is found. We describe their common components.

Neighbourhood structures

Different neighbourhood structures are used when dealing with hard and soft constraints. When only hard constraints are considered, we use two neighbourhoods:

1. The neighbourhood $\mathcal{N}_1(a)$ is defined as the set of assignments obtained from a by moving a single event to a different room and timeslot. More formally, given a with an event E_k assigned to R_i in T_j and a free cell X_{lm} in the assignment matrix, a neighbour assignment a' is obtained from a by setting $X_{ij} = -1$ (*i.e.*, making cell X_{ij} free), $X_{lm} = E_k$, and all other positions are left unchanged.

2. The neighbourhood $\mathcal{N}_2(a)$ is defined as the set of assignments obtained from a by swapping timeslots and rooms of two events. More formally, given a with two events E_h and E_k assigned respectively to R_i in T_j and to R_l in T_m, a neighbour assignment a' is obtained from a with $X_{ij} = E_k$, $X_{lm} = E_h$, and all other position unchanged.

In practice, similar neighbourhood restrictions as for the GCP are used. Thus a local search in $\mathcal{N}_1$ considers only assignments obtained by moving an event involved in at least one hard constraint violation, and a local search in $\mathcal{N}_2$ considers only assignments obtained by swapping pairs of events of which at least one causes a hard constraint violation.

If, in addition, soft constraints violations are considered in the local search, we define the two neighbourhoods $\mathcal{N}'_1 \subseteq \mathcal{N}_1$ and $\mathcal{N}'_2 \subseteq \mathcal{N}_2$, obtained by considering only neighbours that do not introduce violations of the hard constraints. We also use two additional neighbourhoods that do not affect hard constraints.

3. The neighbourhood $\mathcal{N}_3(a)$ is defined as the set of assignments obtained from a by swapping all events assigned to two timeslots. Formally, given two timeslots T_i and T_j of an assignment a, a neighbour assignment a' is the assignment a with the two columns of the assignment matrix $X_{\cdot i}$ and $X_{\cdot j}$ swapped, and all other positions unchanged. The feasibility of the assignment is preserved since no hard constraint is involved.

4. The neighbourhood $\mathcal{N}_4(a)$ is defined as the set of assignments obtained from a by a Kempe chain interchange (see Section 4.6, on page 105, and for an application in timetabling Thompson and Dowsland, 1998). A feasible assignment a corresponds to a partition of events in independent sets, *i.e.*, the timeslots. A Kempe chain K is a connected component in the graph S induced by the events that belong to two different independent sets T_i and T_j, $i \neq j$. A Kempe chain *interchange* produces a new feasible assignment by swapping the events belonging to K. For an example, consider Figure 6.2 of page 225. If we assume that event E_{15} cannot be scheduled in the same timeslot as both events E_{10} and E_{12}, then a Kempe chain interchange of $K = \{E_{15}, E_{10}, E_{12}\}$ moves E_{15} to timeslot T_i and E_{10} and E_{12} to timeslot T_j. We avoid the case $K = T_i \cup T_j$, because the interchange would correspond to a swap already obtainable by neighbourhood $\mathcal{N}_3$. After a Kempe chain interchange, rooms are re-assigned by the matching algorithm defined in Section 6.6.3. In the example, rooms are re-assigned for all events in timeslots T_i and T_j. If no feasible matching of rooms can be found, the current Kempe chain interchange would break feasibility, and, therefore, it is discarded.

Neighbourhood exploration and evaluation function

In all cases, the local search is a *stochastic first improvement local search* in which the examination of the neighbourhood is randomised and the first non worsening neighbour is accepted. In the neighbourhoods $\mathcal{N}_1$, $\mathcal{N}_2$, $\mathcal{N}'_1$, and $\mathcal{N}'_2$ we decide to accept side walk moves,

given the conjectured presence of large plateaux (see also discussion in Section 6.7.4). Side walk moves are not accepted, instead, in $\mathcal{N}_3$ and in $\mathcal{N}_4$, because we observed that in those cases they slow down significantly the search.

In neighbourhood structures $\mathcal{N}_1$, $\mathcal{N}_2$, $\mathcal{N}_1'$, and $\mathcal{N}_2'$, a list of randomly ordered events is created. The list is scanned in the order and for each event all possible moves are tried. The first non worsening move found in this process is accepted and performed. The search continues in the next local search iteration from the successive event in the list. When the whole list of events has been scanned, and no improvement has been found, the procedure ends and the current assignment is recognised as a *local optimum* (side walk moves do not account for improvements). In $\mathcal{N}_1 \cup \mathcal{N}_2$ and $\mathcal{N}_1' \cup \mathcal{N}_2'$, for each event in the list a move in the first neighbourhood is attempted before than a move in the second neighbourhood. Finally, in the neighbourhood structures $\mathcal{N}_3$ and $\mathcal{N}_4$ the exploration is performed considering all possible distinct pairs of timeslots in random order.

The evaluation function to minimise depends on whether we are trying to minimise the violations of H2 and H3 or the violations of Σ. In this latter case the evaluation function to minimise is the one defined in Section 6.3. In the former case, we define $g(a)$ for the hard constraints H2 and H3 as follows:

$$g(a) = g_2(a) + g_3(a)$$

where, $g_2(a)$ is the number of events which are assigned to a non suitable room, and $g_3(a)$ is the number of events sharing students in the same timeslot.

In all cases possible violations of hard constraints are detected in linear time by means of the data structures. This is chiefly important for the neighbourhoods $\mathcal{N}_1'$ and $\mathcal{N}_2'$. For the soft constraints, instead, only local contributions to the evaluation function are computed.

Comment

An important issue for local search is the connectivity of the search space, *i.e.*, the existence of a path between any two solutions. In highly constrained problems, areas where feasible solutions are located may be disconnected. This is a strong drawback for a local search that tries to minimise soft constraints while maintaining feasibility, because the search process could be trapped in regions not containing an optimal solution. Given the multiplicity of neighbourhoods defined above and the instances of the problem at hand, this seems not to be a problem in our case. Indeed, since an optimal solution always exists, there are always free slots in the matrix X (at least those in the last timeslot of the days because of constraint S1) that can be used to move from one solution to another without breaking hard constraints. This implies that there are at least $5 \times |R|$ free places in the assignment matrix that can be used to move events without breaking hard constraints. The landscape looks, therefore, connected, although it may not be possible to walk through the path of minimal changes to go from one assignment to another. On the instances of the UCTP-C, the higher is the difference between the number of events and the places in the assignment matrix (*i.e.*, its size) the easier the local search can move from one assignment to another one.

Function Hard_Constraint_Solver(I_k, a) %$I_k \in I$ is an instance of the competition, $a \in \mathcal{A}$
$loops \leftarrow 0$;
while a is infeasible and time limit not exceeded **do**
  $a \leftarrow \mathsf{Iterative\_Improvement}_{\mathcal{N}_1 \cup \mathcal{N}_2}(a)$;
  if a is infeasible and $loops > 5$ and time limit not exceeded **then**
    $a \leftarrow \mathsf{Tabu\_Search}_{\mathcal{N}_1}(a)$;
  end
  $loops \leftarrow loops + 1$;
end
return a

Algorithm 6.2: High-level pseudo-code for the hard constraint solver.

6.6.6. Hard constraint solver

Algorithm 6.2 sketches the hard constraint solver. First, only $\mathsf{Iterative\_Improvement}_{\mathcal{N}_1 \cup \mathcal{N}_2}$ is applied. As mentioned in the previous section, this procedure may end in a solution which is not precisely a local optima. It was observed experimentally that if re-started the local search finds other improvements (this is due to the fact that it accepts also side walks moves which may cause to move through the plateau and find an exit). Empirically we determined that after 5 re-starts this capacity of producing improvements diminishes drastically (clearly, this observation is restricted to the instances of the competition). Hence, if after 5 runs of local search a feasible solution is still not found, a second phase begins in which local search is alternated with Tabu Search. $\mathsf{Tabu\_Search}_{\mathcal{N}_1}$ implements a best improvement strategy and ends after a number of iterations in which no improvement in the evaluation function is found. The local search at the beginning provides a rapid descent towards assignments that violate only few hard constraints and therefore exhibit a neighbourhood which may be effectively explored by Tabu Search.

More in detail, $\mathsf{Tabu\_Search}_{\mathcal{N}_1}$ is inspired by $\mathsf{TS}_{\mathcal{N}_1}$ for graph colouring (see Chapter 4). A neighbour is forbidden if in it an event receives the same timeslot he had assigned less than tl steps before. The tabu length is $tl = ran(10) + \delta \cdot n_c(a)$ where $ran(10)$ is a random number in $\{0, \ldots, 10\}$, $n_c(a)$ is the number of events in the current assignment involved in at least one hard constraint violation and δ is a parameter. An aspiration criterion accepts a tabu move if it improves the best known assignment. The procedure ends, when the best assignment is not improved for a given number of steps. We fixed this number to $0.4 \cdot |T| \cdot |E| \cdot |R|$ and δ to 0.6.

After a feasible solution has been produced, its quality with respect to the soft constraints, is assessed by means of Look_Ahead_Iterative_Improvement, which consists in a single repetition of local search in, successively, $\mathcal{N}_1'$ and $\mathcal{N}_2'$. This procedure is very fast.

6.6.7. Soft constraint optimiser

An outline of the soft constraint optimiser for minimising the soft constraint violations is given in Algorithm 6.3. It consists of two parts. First, several local search operators (Iterative_Improvement) based on different neighbourhoods are applied until none of them can improve further the assignment, giving rise to a Variable Neighbourhood Descent. Next, Simulated Annealing ($\mathsf{Simulated\_Annealing}_{\mathcal{N}_1' \cup \mathcal{N}_2'\_\mathsf{withMatching}}$) is applied until the end of the available time. Usually, Simulated Annealing reaches good performance

Function SoftConstraintOptimiser(I_k, a) %$I_k \in I$ is an instance of the competition, $a \in \mathcal{A}$
repeat
  $a \leftarrow$ Iterative_Improvement$_{\mathcal{N}'_1}(a)$;
  $a \leftarrow$ Iterative_Improvement$_{\mathcal{N}'_1 \cup \mathcal{N}'_2}(a)$;
  $a \leftarrow$ Iterative_Improvement$_{\mathcal{N}'_1\text{_withMatching}}(a)$;
  $a \leftarrow$ Iterative_Improvement$_{\mathcal{N}'_1 \cup \mathcal{N}'_2\text{_withMatching}}(a)$;
  $a \leftarrow$ Iterative_Improvement$_{\mathcal{N}_3}(a)$;
  $a \leftarrow$ Iterative_Improvement$_{\mathcal{N}_4}(a)$;
until no improvement found;
$a_{best} \leftarrow$ Simulated_Annealing$_{\mathcal{N}'_1 \cup \mathcal{N}'_2\text{_withMatching}}(a)$;
return a_{best}

Algorithm 6.3: High-level pseudo-code for the soft constraint optimiser.

when long computation time is allowed. Here, we start Simulated Annealing from a good solution rather than from a random one and this seems to help in getting good quality solutions quickly. The experimental results indicated this as the best choice in our case.

It was observed that after the application of the Variable Neighbourhood Descent phase, finding further improvements becomes very difficult. It appeared therefore reasonable in Simulated Annealing to trade speed in terms of the number of visited solutions in favour of a deeper exploration of the neighbourhood. The matching algorithm described, in Section 6.6.3, is then introduced for the assignment of rooms and experimental results confirmed that this is a good choice. It is applied jointly with the neighbourhoods $\mathcal{N}'_1$ and $\mathcal{N}'_2$. Every time a change in a timeslot is proposed by introducing a new event or by swapping two events the rooms are re-assigned. The contribution of this mechanism is that new changes are made available which were previously hidden by the fact that few suitable rooms for the incoming event were already occupied. Note that this mechanism is very similar to the re-assignment of colours to vertices in the set T-colouring problem.

The instantiation of Simulated Annealing (SA) is based on the procedure introduced in Section 2.4.3. The following are the components that deserve further comments.

Neighbourhood examination scheme: For each event chosen from a randomly ordered list all timeslots are considered in lexicographic order. First, the possibility of inserting the event in the destination timeslot with the aid of Matching without breaking feasibility is checked (corresponding to a move in neighbourhood $\mathcal{N}'_1$ with matching of rooms). If the insertion is feasible, it is accepted with probability given by the SA acceptance criterion (Equation 2.2, page 25) and the procedure restarts. If it is unfeasible or not accepted, swaps with the aid of Matching with all events assigned in the destination timeslot are tried (neighbourhood $\mathcal{N}'_2$ with matching of rooms). If a swap is feasible it is accepted with probability given by the SA acceptance criterion. For the sake of clearness this scheme is also sketched in Algorithm 6.4.

Note that all events are processed and no restriction on their choice is active.

Initial Temperature: For determining the initial temperature a sample of 100 neighbours in $\mathcal{N}'_1 \cup \mathcal{N}'_2$ of the initial assignment passed to SA is considered. The initial temperature is then determined by the average value of the variation in the evaluation function multiplied by a factor κ ($\kappa > 0$).

Temperature schedule: We used a non monotonic temperature schedule obtained by the interaction of two strategies: a standard geometric cooling and a temperature re-

```
Procedure Simulated_Annealing_{N'_1 ∪ N'_2}_withMatching;
while still time do
    start: select randomly an event e_1 and let t_1 be the timeslot of e_1;
    update Temperature;
    for t_2 from 1 to 45 do
        try to insert e_1 into t_2, t_1 ≠ t_2, by applying Matching on t_1;
        if the insertion is feasible then
            if the SA probabilistic criterion accepts the move then
                remove e_1 from t_1 and insert it in t_2; continue from start;
            end
        end
        for all events e_2 in t_2 do
            try to swap e_1 and e_2 by applying Matching on t_1 and t_2;
            if the swap is feasible then
                if the SA probabilistic criterion accepts the move then
                    swap the events e_1 and e_2; continue from start;
                end
            end
        end
    end
end
return a_best
```

Algorithm 6.4: An algorithmic sketch of Simulated_Annealing$_{\mathcal{N}'_1 \cup \mathcal{N}'_2}$_withMatching.

heating. In the standard geometric cooling the temperature $\mathcal{T}_{i+1}$ is computed from $\mathcal{T}_i$ as $\mathcal{T}_{i+1} = \alpha \times \mathcal{T}_i$ where α is a parameter called the cooling rate ($0 < \alpha < 1$).

The number of iterations at each temperature, *i.e.*, the *temperature length*, is kept proportional to the size of the neighbourhood, as suggested by the majority of SA implementations (Johnson et al., 1991). We set this value equal to $\rho \cdot |E| \cdot |R| \cdot |T|$, where ρ is a parameter, and count as iteration each attempt to make a change in the assignment, independently from the fact that the change is very soon discarded because it breaks feasibility, or that it is accepted.

When no improvement is found for a number of steps given by η times the temperature length, where η is a parameter, the temperature is increased by adding the initial temperature to the current temperature.The assignment that we consider for testing improvements is the best since the last re-heating occurred.

The race indicated that the following was the best set of parameters: $\kappa = 0.15$, $\rho = 10$, $\alpha = 0.95$, $\eta = 5$.

A flowchart resuming the overall algorithm is given in Figure 6.3.

6.7. Analysis of the algorithm

6.7.1. Benchmark comparisons

The International Timetabling Competition gave us the opportunity to compare the final algorithm output from our development methodology with the best results that current

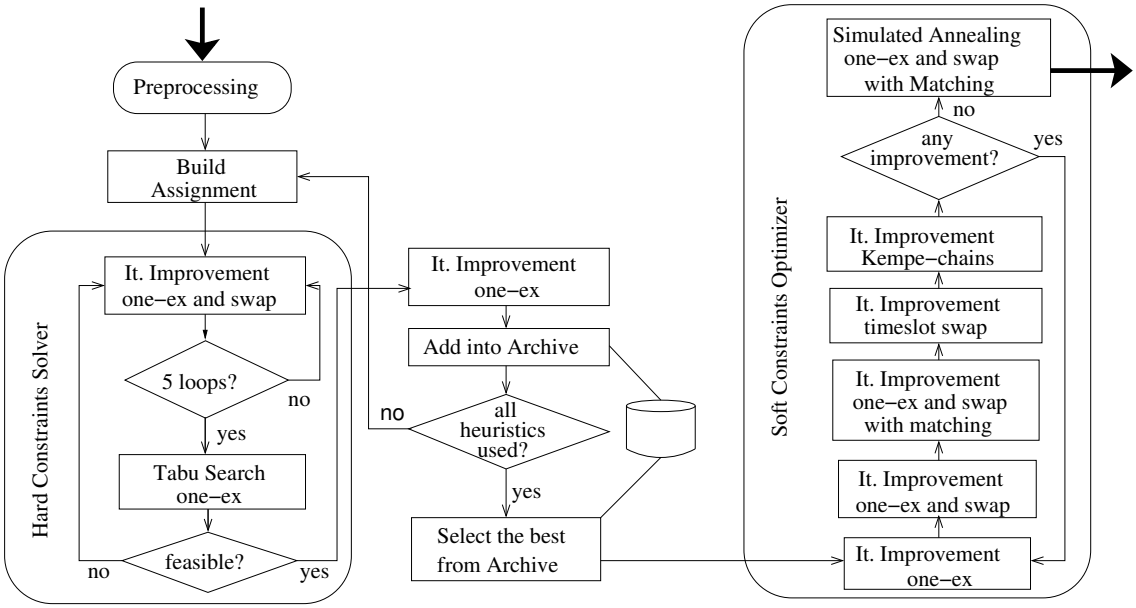


Figure 6.3.: The flowchart of the hybrid algorithm. "Build assignment" constructs an initial tentative timetable using a selected heuristic h_i. Next, local search optimisation through Iterative Improvement and Metaheuristics (Tabu Search and Simulated Annealing) is used to make the timetable first feasible, and then good with respect to soft constraints.

researchers on timetabling are able to produce. According to the rules and the evaluation methodology of the competition presented in Section 6.3, our algorithm scored the lowest penalty value and therefore attained the best rank.

The evaluation methodology of the competition is influenced by the submissions of all participants. We do not reproduce this evaluation here; the results are available on the official web site of the competition.[8] We limit ourselves to compare our results with those of the official winner who scored a penalty value clearly lower than all other competitors. For a description of the algorithm we refer the reader to Kostuch (2003) or Kostuch (2004). The best results attained by the two algorithms are compared in Table 6.3. Our algorithm obtains a better solution on 18 out of 20 instances and the exact binomial test recognises a significant difference between the two algorithms.

For the sake of fairness, it has to be said that the official winner of the competition had much less runs available to choose the best submission (P. Kostuch, priv. comm.). Yet, we deem that this fact gives further credit to the effectiveness of our experimental methodology. With a brute force approach we could not have collected either one single run per each of the 20 instances or we should have drastically reduced the number of tested configurations, possibly leaving out the best one.

The competition proved, therefore, the effectiveness of our approach. Nevertheless, it is also important to emphasise that even without the competition the experimental approach adopted in our development methodology brings enough warranties on the quality of the outcome algorithm. The statement, made possible by the race, "the algorithm is the best out of 1185 other candidates" is indeed a strong argument in favour of its reliability.

[8] B. Paechter, L.M. Gambardella, O. Rossi-Doria. "International Timetabling Competition". 2003. http://www.idsia.ch/Files/ttcomp2002/ (October 2004).

Instance	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
Official Winner	45	25	65	115	102	13	44	29	17	61
Our Subm.	57	31	**61**	**112**	**86**	**3**	**5**	**4**	**16**	**54**
After Optim.	**45**	**14**	**45**	**71**	**59**	**1**	**3**	**1**	**8**	**52**
Instance	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*
Official Winner	44	107	78	52	24	22	86	31	44	7
Our Subm.	**38**	**100**	**71**	**25**	**14**	**11**	**69**	**24**	**40**	**0**
After Optim.	**30**	**75**	**55**	**18**	**8**	**5**	**46**	**24**	**33**	**0**

Table 6.3.: Results on the 20 instances of the International Timetabling Competition. The first two rows are the official submissions to the competition. A number in bold face indicates that our algorithm wins. The third row reports the results out of 50 runs per instance attained by an optimised version of our code.

After the submission we polished and optimised the code of the algorithm and we run it again 50 times on each instance. In Table 6.3 we present the best results attained while in Figure 6.4 we show the distribution of the results by means of box-plots. The two following considerations can be drawn: (i) some instances appear better solvable than others because their solutions are closer, in terms of quality, to the optimal solutions (in particular instance 20 is the easiest followed by instances 6, 8, 7); (ii) the harder the instance the higher is the variance of the range in the distributions of the evaluation function.

Recently, Kostuch (2004) has published new results for this set of instances, which are better than the results of our optimised algorithm. Our intention is not to defend our algorithm rather to sustain the appropriateness of the systematic development methodology introduced. We do believe there can still be room for improvements in both the algorithms, but within the strict rules of the competition, above all within its fixed time limit (in fact, a very realistic circumstance), the use of the experimental methodology proved to be the determinant factor of our success.

6.7.2. Contribution of algorithm components

In this section, we report about a post competition analysis aimed at the understanding of the real impact of each component on the final performance of the algorithm. As a logical consequence, we answer to the plausible concern whether all components are really needed or whether the algorithm could be simplified.

A first relevant observation is obtained by profiling our code. Construction heuristics and local searches on hard and soft constraints, consume only 10% of the total time allowed for a run of the algorithm, while the remaining 90% is used by Simulated Annealing. This may suggest that possible further efforts in optimising the code should be directed in the first place to that second part of the algorithm.

Construction heuristics

In order to understand if some heuristics are more useful than others, we run the algorithm several times with different random seeds until the procedure Select_Best_Assignment in Algorithm 6.1 is completed and we record the frequency with which each

construction heuristic provides the initial solution from which a_{best} is derived. If some heuristics were never selected we could discard them, while knowing which heuristics are more used could help us in exploiting features of the problem. Unfortunately, these are not the cases of our experiment.

In Figure 6.5, left, we report the histogram of the number of times each heuristic produced the best assignment. Results are aggregated from a set of 50 runs for each of the 20 instances. The Kolomogorov Smirnov test (Conover, 1999) indicates that there is not enough significance to reject the hypothesis that the observed distribution is sampled from a discrete uniform distribution. Visual evidence of this fact is given in Figure 6.5, right. This is an interesting result: apparently, each heuristic has the same probability of leading to a_{best}. The same analysis repeated on each single instance, confirms that, in all the instances except two, the distribution of the selection frequency remains not significantly different from a uniform distribution. Note that, since all heuristics are deterministic and generate an assignment which is still infeasible, the stochasticity of the selection process is due to the hard constraint solver and the look ahead local search that follows.

However, the use of the construction heuristics is not ineffective. Indeed, we compared the algorithm that uses the 60 heuristics against an algorithm that generates 60 initial solutions in a semi-random manner. The semi-random construction can be summarised as follows. First all events are assigned randomly to timeslots. Then an exact matching algorithm is applied to each timeslot to assign rooms to events. If in the timeslot there are more events than rooms or if it is not possible to schedule all events in a suitable room, some events do not receive a room. After the matching algorithm has been applied to all timeslots, the events left without room are assigned to suitable rooms in different timeslots, such that the number of violations of constraint H3 is minimised. In the case of ties, priority between events is decided by label order. The comparison shows that the use of the 60 heuristics is preferable, as it produces assignments which are closer to feasibility, *i.e.*, feasibility is reached in less time. This entails that more time is left to the soft constraint optimiser to produce better final solutions.

We may conclude that, the use of the 60 construction heuristics described in Section 6.6.4 is certainly not the main factor of success of our algorithm but they could be substituted only by heuristics which are able to solve the hard constraints better. Heuristics based on the principles of DSATUR in graph colouring appear definitely more promising for this task. Indeed the algorithm of Arntzen and Løkketangen (2003) presented at the competition and based on this concept produces easily feasible solutions for all the instances of the competition. However, preliminary experiments comparing our algorithm modified to use only this heuristic 60 times against the version submitted with the 60 heuristics did not reveal significant differences in their final results.

We recall instead that choosing among a set of initial solutions, rather than using a single one, in order to have low soft constraint violations in the initial solution was indicated as profitable by the race, although this was determined only at the last stage by the binomial test. A set of 60 initial solutions, to which corresponds the use of $1/10$ of the total running time, apparently is a good compromise between searching for good initial solutions and leaving enough time to the soft constraint optimiser.

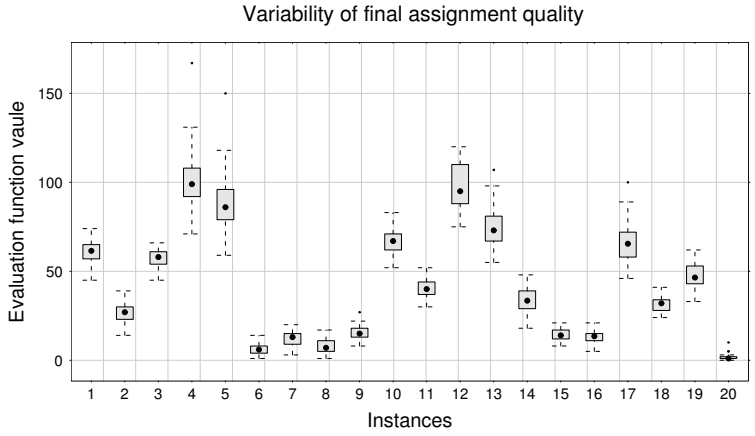


Figure 6.4.: Box-plots of the distributions of the solution quality on the 20 instances of the competition with the optimised code.

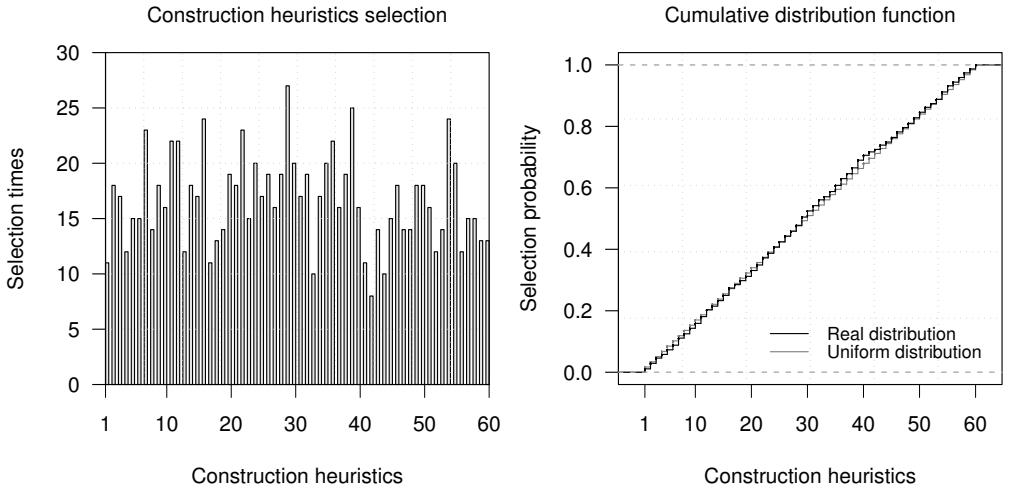


Figure 6.5.: Diagnostic plots in support of the observation that heuristics are selected with equal probability. The left plot gives the number of times each heuristic led to the best feasible solution. The right plot gives the visual comparison between the cumulative observed frequency and a discrete uniform distribution.

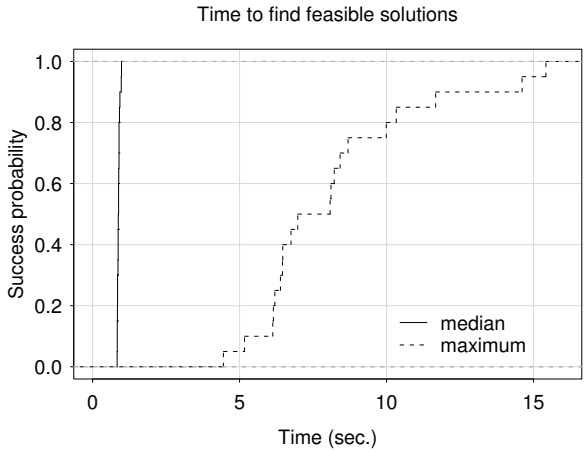


Figure 6.6.: Empirical cumulative distribution of the median (solid line) and maximal (dashed line) computation time needed to find a feasible solution. Results are derived from 30 runs on each of the 20 instances.

Hard constraint solver

Once a single initial assignment has been constructed by one of the construction heuristics, making the assignment feasible is not a hard task. To exemplify this, we re-run our algorithm 30 times on each instance and stopped it when it found a feasible solution. The algorithm used only once a single construction heuristic (events assigned according to the maximal number of adjacent events and rooms and timeslots in label order) and then run the procedure Hard_Constraint_Solver. To represent the distribution of the time needed to reach a feasible assignment over all the instances, we consider two statistics, the median and the maximal time for each instance. In Figure 6.6, we plot the empirical cumulative distribution of the computation time for finding a feasible solution (the time limit imposed by the competition was 1074 seconds). [9] Considering the maximal time, represented by the dashed line, we observe that for any instance 15 seconds are sufficient to find a feasible solution. Considering the median time (solid line), a feasible solution for a given instance is produced already after only 1 second in 50% of the trials. The insight that solving the hard constraints is not a hard task was already suggested by the easiness of the graph colouring instances. In practice, this allowed us to invest more time to improve the part of the algorithm which deals with minimising the soft constraint violations.

Simulated Annealing

Table 6.4 shows the improvement in solution quality brought by the application of Simulated_Annealing$_{\mathcal{N}_1' \cup \mathcal{N}_2'_\mathsf{withMatching}}$. The improvement is relevant, and without the SA phase, results in the competition would have been poor.

A typical profile of the temperature and the evaluation function value for the SA is given in Figure 6.7. We observe that the re-heating feature is used on average 2-3 times and that its effect is profitable, because in many runs the best solutions are obtained close to the time limit. On average, SA performs 70×10^4 iterations per second for a total

[9] This experiment was run on a Pentium III, with a 700 MHz CPU, 256 KB cache, and 512 MB RAM.

Instance	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
Before SA	199	150	208	385	388	286	101	140	120	193
After SA	77	42	78	162	120	14	18	20	31	82
Instance	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*
Before SA	210	228	260	263	229	115	356	156	273	157
After SA	58	143	109	49	5	24	121	46	68	4

Table 6.4.: The contribution of Simulated Annealing in terms of evaluation function values. Reported are the median values on 96 trials per instance.

of about 40×10^7 iterations. We recall however that we count as iteration every single attempt of moving an event in the assignment matrix which may be also an infeasible move. The cost of a single iteration, therefore, varies considerably depending on how soon the proposed change is refused or accepted.

6.7.3. Qualified run time distributions

In order to analyse the behaviour of the algorithm over time we plot the qualified run time distribution introduced in Section 3.8.1. We report them in Figure 6.8. They correspond to different solution quality levels on instance 20, which was the easiest instance to solve. We observe that an assignment of cost at least 3 will be always reached in the allowed time limit, while an optimal assignment will be found in about 30% of the cases. We may conclude that longer run time could be profitable, as the slope of the distributions is positive and improvements can still be expected. As a consequence of this observation, we left our algorithm run longer also on instances 6, 7, 8 and, indeed, we reached optimal solutions for these cases.

We check whether it is correct to leave the SA run on the same solution for all the time available or whether it stagnates and restarts of the algorithm would be opportune (Hoos and Stützle, 1999). To this end, we compare the empirical distributions with exponential distributions of parameters $\theta = 0$ and $\lambda = 1/\bar{t}$. The fit is not satisfactory. This fact motivates the search for possible *cut-off* points. In Figure 6.8 we report the exponential distributions with inclination such that they touch the empirical distributions in points where the steepness of the empirical QRTDs becomes lower than the one of the exponential distributions. Those are the points where a restart could be opportune. Nevertheless, in our case it arises that after those points the steepness of the emprical QRTDs approximates the one of the exponential distributions. Given that a restart would imply a new initialisation phase to be finished before seeing again a strong rise of the QRTDs as in their first part, then we may conclude that a restart of the algorithm is not needed.

6.7.4. Landscape Analysis

From the description of the algorithm it emerges clearly that it is strongly based on local search components. The characterisation of the search landscape may help therefore in the understanding of its behaviour. In particular, we look for models based on the features of the search landscape which explain the differences in the hardness of the instances. A similar study is described for the Job Shop Scheduling problem by Watson

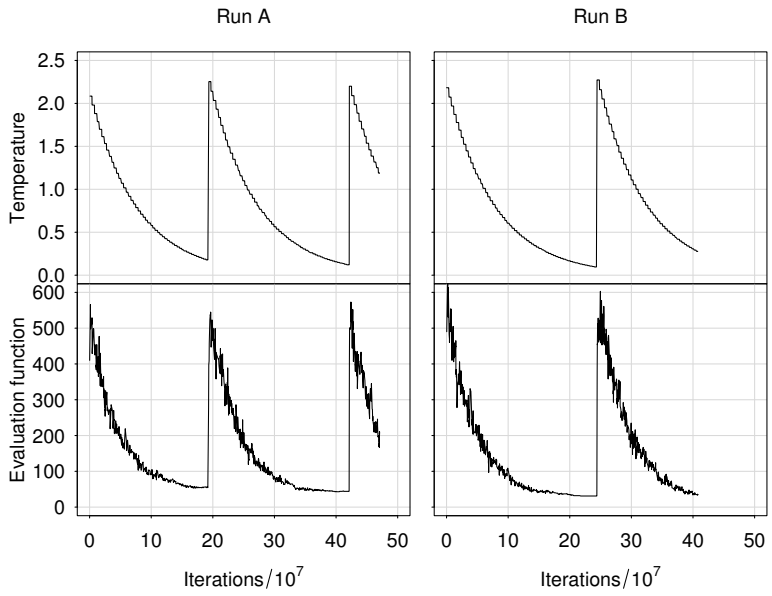


Figure 6.7.: Profile of temperature (upper graphs) and evaluation function value (lower graphs) for two representative runs (left and right parts) of Simulated Annealing on instance 14. The number of iterations is reported on the x-axis. The runs "A" and "B" ended respectively with 43 an 31 soft constraint violations and performed a different number of iterations. Since the CPU time for the two runs is fixed, differences in the total number iterations are due to the stochasticity inherent to the algorithm.

et al. (2003). Yet, given the hybrid nature of the algorithm under study, the search landscape is not univocally defined but changes with each phase and with each different neighbourhood. We decided therefore to restrict ourselves to consider only the search landscape determined by the one-exchange neighbourhood, *i.e.*, $\mathcal{N}_1$, when the evaluation function is determined solely by the soft constraints. After all, our main interest is on the landscape when solving the soft constraints because we already showed that finding a feasible solution is easy.

Every instance has an optimal solution to which corresponds a value of 0 in the evaluation function. It seems therefore reasonable to use as indicator of the hardness to solve an instance the quality of the final solution found. Precisely, we define as *cost of solving an instance* the median value of the final solution qualities found in 50 runs of our algorithm. This corresponds to the median value of each distribution reported in Figure 6.4. In this way we assume that the closer this value is to zero the easier it is solving the instance. Alternative measures of the hardness are the number of iterations or the time needed to reach an optimal solution or a given level of the solution quality.

Before proceeding in the analysis, we also define the *distance* between two solutions in the search space. To this end, we assume a further simplification of the problem by ignoring the room assignment and including the possibility of visiting an infeasible assignment. The consequence of this simplification is that the distance between solutions is in fact a lower bound of the real distance because the path made solely of one exchanges may not be feasible due to the impossibility at some steps to find a feasible room assignment.

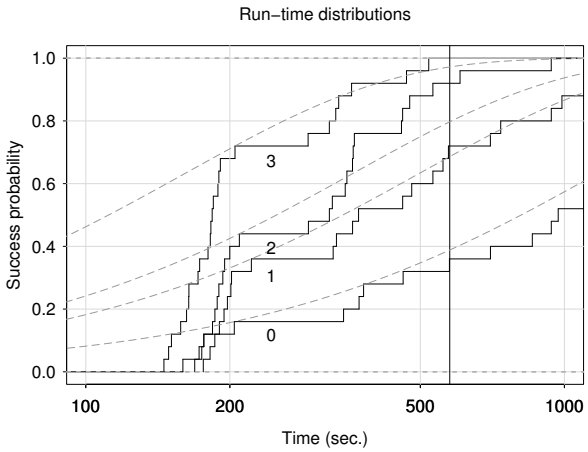


Figure 6.8.: Run-time distributions for four different solution quality levels on instance 20. At each time a point in the curve represents the empirical probability to find a solution of the specified quality. Empirical data are taken from 25 runs. The superimposed dotted lines represent exponential distributions while the vertical line is the total time limit on a machine with processor AMD Athlon, 1 GHz CPU, 256 KB cache, and 512 MB RAM.

In computing the one-exchange distance, we consider two assignments to be equal if only days are permuted. Hence, the distance corresponds to the *partition distance* in graph colouring. More precisely, we compute the distance between two assignments a_1 and a_2 in the following way. For each day of assignment a_1 we compute the similarities with each day of assignment a_2 and we store these values in a 5×5 matrix called *matrix of costs* (5 is the number of days). The number of similarities is obtained by a matched pairwise comparison of timeslots in the two days (*i.e.*, between timeslots with the same index in the two days), and corresponds to the number of events that are present in both timeslots. Next, we look for the permutation of days in the assignment a_2 that would yield the maximal total number of similarities between a_1 and a_2. This is achieved by solving the linear assignment problem on the matrix of costs Gusfield (2002). To this end, we use, the algorithm of Jonker and Volgenant (1987).[10] This is not a polynomial time algorithm but it is very fast for the small instance size (5×5) that we have to solve.

Quality of local optima.
The first feature that we consider is the quality of local optima. Intuitively, the worse the quality of local optima the harder the problem is to solve. In our case, we define a local optimum to be the assignment reached after the variable neighbourhood descent in the soft constraint optimiser, as defined in Section 6.6. Note, however, that, the solution after the variable neighbourhood search is not necessarily a strict local optimum.

For each instance, we generated 200 local optima and we considered the medians of the distribution of their quality. In Figure 6.9, we show scatter plots of the median quality of local optimum versus the instance cost. A linear regression on these points attains an r^2 of 0.503 and there is enough statistical significance to reject the hypothesis of independence between the two variables. According to this result, the quality of local optima after the variable neighbourhood descent explains to some extent the final difficulty of the instance. Consequently, it could be possible to predict, with a certain error, the final

[10] MagicLogic Systems Inc. "Linear Assignment Problem - software downloads." http://www.magiclogic.com/assignment.html. (April 2005.)

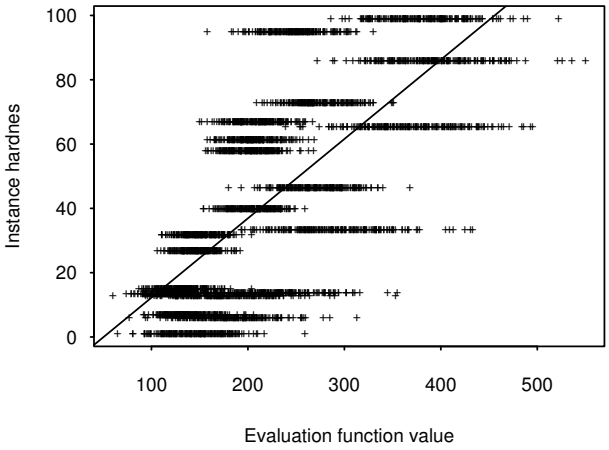


Figure 6.9.: Scatter plot of the median costs of local optima versus the cost of an instance (median costs of final solutions); a least-squares fit line is superimposed.

solution quality after just a few seconds of running the algorithm.

We investigated also the correlation between the quality of the initial feasible assignment chosen after the look ahead local search and the final solution quality. A correlation between these two features would indicate a certain importance of starting from a good assignment and could justify the selection of the best candidate among the initial set of candidates. However, in this case the null hypothesis that the two variable are independent could not be rejected.

Local optima localisation

From the same sample of local optima of the previous paragraph we considered: (i) the minimal distances of each local optimum from another local optimum and (ii) the distances of each local optimum from the unique global optimum available. Clearly, at point (ii) the distances may badly represent the actual distance from global optima because there may be other global optima much closer. However, the unique global optimum available may be part of a larger plateau. Therefore, in order to determine the smallest distance, we apply to it a Tabu Search on the $\mathcal{N}_1'$ and $\mathcal{N}_2'$ neighbourhood structures that minimises the distance while keeping fixed the value of the evaluation function. The minimal distance of the local optimum from the global optimum is then the minimal distance found in this search.

In Figure 6.10, we show the distributions of these distances when normalised by the number of events. It shows clearly that solutions are all very far away from each other. Indeed, both distributions peak at more than 0.9 normalised distance which means that almost all events must be relocated to pass from one solution to the other. Usually, the distance of local optima from global optima is a good indicator of the hardness of solving an instance (Watson et al., 2003). Unfortunately, given that we had only one global optimum available and given the similarity between the curves in the figure, it is not possible to verify the truth of this statement in our case.

Finally, we mention that we also checked the relation between the minimal distance of local optima from global optimum and the quality of local optima. Intuitively, assignment closer to optima should have better quality, but this was not confirmed by our experiments, where no significance arose to reject the hypothesis that the two measures

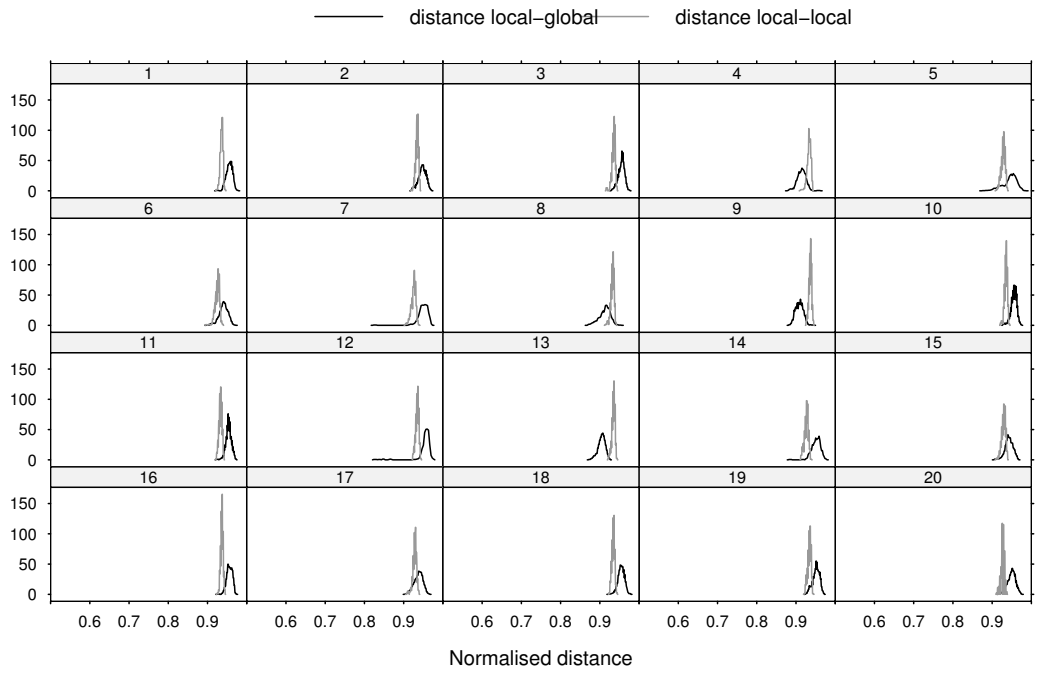


Figure 6.10.: The distribution of distances within local optima and between local optima and global optima on each of the 20 instances. The number of local optima considered is 200. The normalised distance ranges from 0 to 1, although in the plot we show only the interval $[0.5, 1]$. The distributions' densities are obtained by smoothing the empirical distributions.

are independent.

Plateaux size in local optima and global optima

In order to determine the size of the plateaux we implemented a local search procedure that tries to explore the plateau by modifying as much as possible an assignment, while maintaining fixed the value of the evaluation function. In order to avoid visiting twice a same assignment, this procedure keeps an archive of the visited assignments. Contrary to a Tabu Search approach, it records the whole assignment in order to be sure not to count solutions twice. This makes the search computationally quite expensive.

This local search is triggered when a local optimum is reached (hence, again, after the variable neighbourhood descent in the soft constraints optimiser), and we let it run for 10 times the usual time limit. If the exploration of the plateaus finishes before the time limit is exceeded, a new local optimum is generated by restarting the whole procedure. Note that this algorithm does not yield the exact size of the plateau because its result depends on the trajectory. Nevertheless, for a qualitative indication of whether plateaux are large or not, this algorithm may suffice.

In Figure 6.11, we report by means of bar-plots the size of the largest plateaus found by the algorithm during one run per instance. We also report the same analysis for the available global optima. We observed that in some instances the time was not even enough to complete the exploration of one single plateau. With some few exceptions, the size of the plateaux is very large. The largest counted 200 thousand assignments. Surprisingly, when plateaux are small for local optima, the same holds also for global optimum. The size of plateaux appears therefore a characteristic of the instance. We were, however, un-

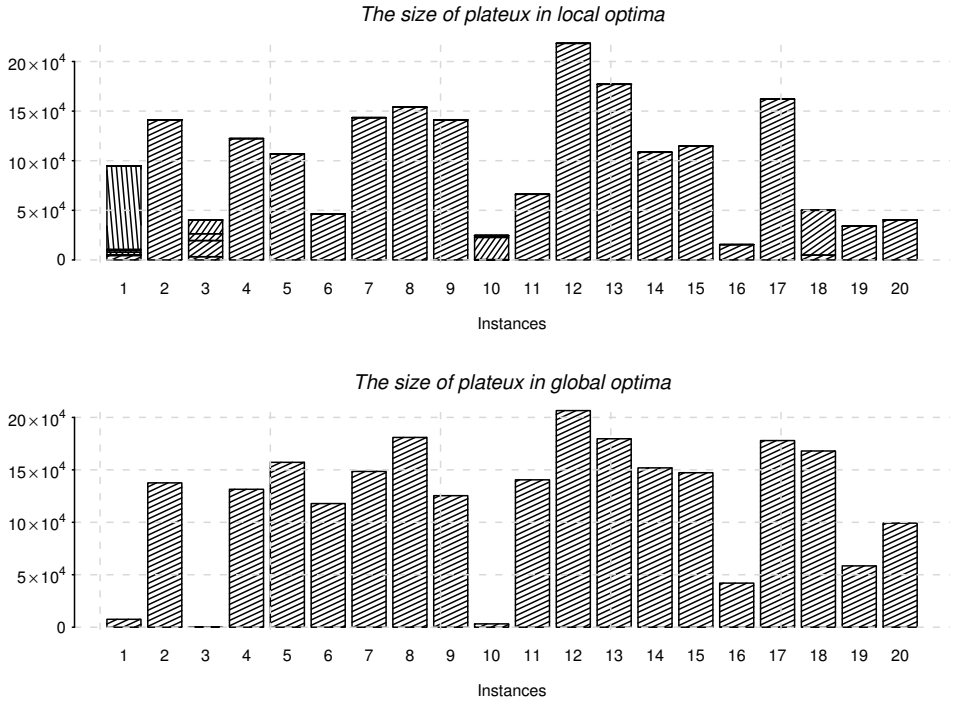


Figure 6.11.: A qualitative view of the size of plateaux in local optima and global optima. The size has to be intended as solely indicative.

able to find any correlation between the size of plateaux and other characteristics of the instance. Hence, this phenomenon remains unexplained.

The conclusion from Figure 6.11 is that plateaux are in general large. This fact justifies, in part, our choice of accepting side walk moves but controlling that their acceptance does not cause the search to remain trapped in a single plateau (as described in Section 6.6.5 each local search accepts the first non worsening move but ends if the last $|E|$ moves were all side walks).

In each single run of our algorithm the number of side walk moves is, on average, 9 times greater than the number of improving moves. Side walk moves are then positively correlated with the size of largest plateaux in the instances. In contrast, no significant correlation exists between the number of side walk moves and the hardness of the instance or the final solution quality.

Search space connectivity

We mention that on instance 20, where we had available more than one global optimum, we succeeded in finding a path from one solution to another. This was achieved by a local search in $\mathcal{N}_1' \cup \mathcal{N}_2'$ that minimises the distance between the current solution and a target solution, *i.e.*, an other global optimum, and that is not allowed to break feasibility. The success of this experiment gives further credit to the conjecture that the search space in the neighbourhood $\mathcal{N}_1' \cup \mathcal{N}_2'$ is indeed connected.

Correlation with $\tilde{f}(i)$			
# events	0.096	AV(students/event)	−0.172
# students	−0.169	SD(students/event)	−0.112
# rooms	−0.311	AV$(d(G'_T))$	**0.456**
AV(rooms/event)	−0.370	SD$(d(G'_T))$	0.153
AV(events/student)	0.253	$\widehat{\chi}(G_T)$	0.036
SD(events/student)	−0.174	$\widehat{\chi}(G'_T)$	**0.469**

Table 6.5.: Correlation between quantitative variables to explain the hardness of an instance represented by $\tilde{f}(i)$. The number in bold face indicates where independence is significantly rejected.

6.7.5. Further analyses

Modelling the hardness of an instance

It may be useful in a real context to understand which are the features of an instance, which are known *a priori*, that have an influence on the final solutions produced by our algorithm. In a university, once these features are known, it may be possible to negotiate them with other departments, or to undertake other preventive measures in order to obtain better timetables in the future. Linear regression techniques could already unveil some important patterns. We attempted this kind of analysis on the results available. Namely, we search for good models to predict the hardness of an instance, indicated again by the median values of Figure 6.4, denoted here as $\tilde{f}(i)$, where i is the instance. Several quantitative variables were considered and their correlation with $\tilde{f}(i)$ is reported in Table 6.5. The highest correlation is for the average event degree of the instances and was statistically significant based on the randomisation test of independence (Cohen, 1995). However, the regression model between average event degree and $\tilde{f}(a)$ gave an adjusted r^2 value of 0.164, which is quite low. Also the chromatic number of the underlying graph presents a statistically significant correlation with the hardness of the instances. However, inducing a variation of the chromatic number of the underlying graph may not be easy due to the fact that it is not merely influenced by the average event degree but it is rather the effect of the global structure of the graph, which is hard to control. Therefore, the practical importance of this finding is limited.

A finer analysis is possible by the use of multiple linear regression techniques. Using a heuristic approach, namely the "stepwise multiple regression", where the significance of predictors is checked by Fisher tests (Cohen, 1995), we arrived to following model which has $R^2 = 0.676$:

$$\tilde{f}(i) = 2.44 \cdot \mathsf{AV}\big(d(G'_T)\big) + 1.20 \cdot |E| + 66.93 \cdot |R| + 1.83 \cdot \mathsf{SD}\big(d(G'_T)\big) - 6.36$$

The analysis of the mutual correlation between terms in the formula indicates that their independence cannot be rejected, and, therefore, their inclusion in the formula is not redundant. We can conclude that the predictors with a significant effect on the quality of final solutions are:

- the average event degree, $\mathsf{AV}\big(d(G'_T)\big)$,
- the standard deviation of the event degree, $\mathsf{SD}\big(d(G'_T)\big)$,
- the number of events $|E|$,
- the number of rooms $|R|$.

Admittedly, the number of available data (only 20 instances) is quite low for considering such an analysis reliable and its inclusion in our discussion has to be regarded mainly as an example rather than oriented to any important conclusion on the UCTP-C. However, a similar analysis on this problem with much more data is developed by Kostuch and Socha (2004) and their results confirm, in general, our findings.

Conflicting constraints analysis
As noted for the set T-colouring problem, in a multi-objective context an important issue is whether objectives are really in contrast with each other or whether optimising one implies automatically optimising also the other. In the UCTP-C we can treat each type of constraint as a different objective defined in Section 6.3 and Section 6.6.5.

We generated 1000 assignments per instance with the semi-random construction heuristic and computed the median over all the instances of the correlation for each pair of constraint types. The correlation is in general very close to zero. The only exceptions are a negative correlation of value -0.20 between H2 and S2, and of value -0.13 between S1 and S2. This indicates that it can be hard to minimise the constraint violations S2, because minimising them we may cause an increase of the violations of the other two types. The negative correlation is reasonable. Indeed, trying not to schedule classes in non suitable rooms can have the effect of spreading them in different timeslots likely causing students to have classes in a row. H2 has, however, higher priority in our algorithm. Under the point of view of our soft constraint optimiser, the slight negative correlation between S2 and S1 is more relevant. It indicates that scheduling at least one timeslot free after a couple of timeslots in which a student already attends a course may cause some event to be assigned to the last timeslot of the day. In contrast, the constraint S3 is not negatively correlated with any other constraint typology. Accordingly, it can happen that satisfying other constraints contributes also to satisfying S3.

It would also be interesting to understand the relative hardness of these constraints but this task requires, unfortunately, a normalisation of results which appears quite problematic.

6.8. General guidelines for the application of SLS methods

The International Timetabling Competition received 24 feasible submissions from all over the world. According to the rules of the competition, our algorithm ranked the best. It is however important to note that 10 out of the 11 best ranked methods where SLS algorithms, that is, metaheuristics and hybridisations thereof. They sometime differ for the methods they use for producing feasible solutions but then, when soft constraints are to be minimised, SLS algorithms are the choice. In particular, the official winner is an algorithm defined in three phases, which as our own algorithm, strongly relies on Simulated Annealing (Kostuch, 2004).

Our hybrid algorithm is based on a framework which consists in the successive application of construction heuristics, Variable Neighbourhood Descent, and Simulated Annealing. Simulated Annealing can then be further alternated with Variable Neighbourhood Descent. This framework is of general applicability. For example, we participated also to the ROADEF'05 competition on a car sequencing problem.[11] In that case, we

[11] Van-Dat Cung. "Challenge ROADEF'2005". March 2005. http://www.prism.uvsq.fr/~vdc/

adopted a similar engineering methodology, although not based uniquely on sequential analysis but also on experimental design, which was possible given the larger amount of time available for the development than for the UCTP-C. Interestingly, the final algorithm, although developed independently from the one discussed here, resulted using the same high level framework (Risler et al., 2004). In that competition the algorithm ranked finally fourth in its category (while it was first after the first phase of the competition).

It may be opportune at this point trying to summarise some guidelines for the application of SLS methods to *highly constrained combinatorial optimisation problems*. These guidelines stem from the studies presented in this thesis and from other complementary experiences of the author. *They are "suggested" by empirical observations and hence strongly dependent on the problem situation in which these observations were collected.* As we saw for very large scale neighbourhoods, there is no guarantee that a result observed on a problem is valid also on another and evidence can be gathered only by testing. In this sense the guidelines below do not have any other pretension than being regarded as mere "*starting hints*".

- Start by devising construction heuristics. This entails a better understanding of the problem and where its difficulties lay. Often, the same rules used in backtracking approaches, most constrained first or least constrained last (as shown for the set T-colouring problem on page 176) work fine.

- Develop good local searches. Recall its four components: (i) solution representation, (ii) evaluation function, (iii) neighbourhood structure, and (iv) search strategy.

  The possible combinations are many and *a priori* there are few elements to guess what is best to do. The ultimate answer should necessarily come from experimental testing. However there may be some general ideas for how to start:

  (i) choose a solution representation that allows to satisfy by construction as many constraints as possible;

  (ii) A prerequisite for the application of SLS algorithms is a fast evaluation of solutions. If this is not possible for the problem at hand, consider the substitution of the evaluation function with a surrogate function, and reduce the number of times the original function is computed. If more than a choice is possible, prefer the one that produces less ties between solutions.

  (iii) develop as many neighbourhoods as are devisable. This part is the most important for the success of the overall SLS algorithm, therefore invest time here. The application of metaheuristics afterwards is often straightforward. The real breakthrough in SLS algorithms is given in the first place by the neighbourhood used in the local search. Usually, inspiration for the neighbourhood structures comes from well known standard problems and may consists of exchanges, insertions, reverses, etc. but *ad hoc* neighbourhoods as we have seen for timeslot exchanges, or colour reassignment in set T-colouring, or room matching are also possible. Combine them intelligently. Usually, heuristic rules may be applied; otherwise the most straightforward approach is to alternate the search among neighbourhoods in the fashion of Variable Neighbourhood Descent. Exploit as much as possible implementation tricks for speeding up the execution of moves.

ROADEF/CHALLENGES/2005/. (April 2005.)

(iv) The way the neighbourhood is explored is crucial for its success. On this aspect the size of the neighbourhood has a strong impact, therefore, try to reduce it by considering only promising neighbours (this corresponds to the use of a candidate list, as denoted by other authors). A way for achieving this is the involvement of problem constraints, possibly controlling that the search space does not become disconnected. A second issue is the evaluation of the neighbours. Here, all possible tricks must be devised for a fast evaluation. A good praxis is the use of auxiliary data structures to compute only the local contribution to the evaluation function. A best improvement strategy works well only with small neighbourhoods. In the other cases, the choice is necessary for a first improvement strategy, or, alternatively, for searching the best in a random sample of the neighbourhood. The acceptance of side walk moves has to be considered carefully. In general, if used sapiently, it can be profitable.

- The application of metaheuristics should come only after all possibilities from the previous two points have been exploited. However, at this point the first rule should be: *keep it simple*. The problem of simplicity is a controversial philosophical issue. Our assertion is based on the following three criteria:
  - smaller number of freely adjustable parameters;
  - higher degree of heuristic guidance with respect to chance-like decisions. In other terms do not make abuse of randomness. Clearly, the extreme case of generating solutions completely at random is a bad approach to solve the problem. Hence, the use and test of rules is to be preferred because it enables to exploit problem knowledge.
  - lower number of components. Be critical with everything that is added to the algorithm, test empirically, and do not hesitate to remove something if it does not yield a significant improvement.
- General hints on the metaheuristics have much lower evidence than the previous ones. It is hard to forecast the behaviour of a metaheuristic and a valid guideline in this context is: implement more than one and compare them. In new problems, where no benchmarks are available this is also a good way for obtaining an assessment of the results. Each single metaheuristic can then be arbitrarily modified in order to enhance its performance. The comments that follow refer to the basic principles inherent to each metaheuristic as outlined in Chapter 2, Section 2.4.3.

  (i) Iterated Local Search is very easy to apply and it is therefore the first thing to try. It constitutes a good reference algorithm which is often not outperformed by other metaheuristics. There is no indication on the strength of the perturbation which can vary from problem to problem. Sequences of random moves or some move in a different neighbourhood may be used to remain somehow "close" to the local optima found. It is important, however, to avoid that the perturbations be easily undone. At the other extreme there is the complete reconstruction of the solution. Random restart, typically performs poorly, but Greedy Randomised Adaptive Search Procedure and Adaptive Iterated Greedy are appealing when several construction heuristics are available and when they can be randomised.

  (ii) Tabu Search, Guided Local Search, and other probabilistic methods such as the Min-Conflicts heuristic and Novelty$^+$ are alternative methods to implement the same concept, that is, improving the exploration of the search space around the

local optima. Yet, it is often more complicated to control what is happening and to tune these metaheuristics correctly with respect to Iterated Local Search. Tabu Search may be a good choice with small neighbourhoods where a best improvement strategy is feasible. The length of the tabu list may be crucial; linking it to the size of the neighbourhood is a choice which allows to control the fraction of neighbourhood forbidden. The introduction of reactive mechanisms which modify the length of the tabu list during the search increases the complexity of the algorithm and requires additional efforts in tuning. The other methods are instead more appropriate than Tabu Search for large size neighbourhoods. All these methods can be used as local search procedures in Iterated Local Search.

(iii) Simulated Annealing is promising with long run times and deals well with large neighbourhoods. Nevertheless, it is very sensible to its parameters and requires an adequate tuning which is strongly dependent on the instance class and on the time limit. In all cases in which a time limit is not fixed or too short, Simulated Annealing is ineffective because it cannot be properly tuned. Another case where Simulated Annealing is ineffective is when the evaluation function is a surrogate for the effective quality of the solution. In this case Simulated Annealing introduces further noise in the search which is very hard to control. Although somehow in contrast with its principles, we saw that Simulated Annealing seems to perform better when applied from a good starting solution as the one obtained by a Variable Neighbourhood Descent.

(iv) Population based methods such as Ant Colony and Evolutionary Algorithms should be applied when the contributions of the previous methods have been thoroughly exploited and there are still computational resources left (*i.e.*, computation time, or parallel computers). All the heuristics rules that exploit problem knowledge and that have devised for the other methods can be used as part of their heuristic component (ant colony) or recombination component (evolutionary algorithm). The hybridisation with a powerful local search seems also recommendable.

6.9. Discussion

This chapter completed our path from simple theoretical problems to real world applications. We gave evidence of the connections between timetabling and graph colouring problems. Both are *assignment problems with constraints* and ideas for the solution methods can be successfully transferred from one case to the other. Some of these ideas are: the inclusion of constraints in the solution representation, which was suggested in the set T-colouring problem; the transfer of the neighbourhood structures one-exchange, swaps, and Kempe chains; the restriction of the neighbourhood by considering only events that are involved in violations and the use of Tabu Search when the neighbourhood is restricted enough, which arose in the graph colouring; the feasibility of solving exactly small sub-problems, which brought advantages in the set T-colouring problem; the use of sequential heuristics that order events according to their degree of involvement in constraints, similarly to DSATUR for graph colouring problems.

Nevertheless, we argued that the application of SLS methods for solving new complex problems requires a systematic methodology. In the previous chapters we saw that it is

difficult to predict the behaviour of an SLS algorithm before testing it empirically. Therefore, general, prescriptive rules may be misleading. The results on standard problems such as GCP, TSP, MAX-SAT, etc., which have been well investigated in academic studies, are the starting point. Recognising one of these problems in the application at hand allows to restrict the attention to a certain collection of promising components for SLS methods. These components must then be adapted to the current problem. As noted in the previous chapters the elements to forecast *a priori* the behaviour of an SLS component or algorithm are very few. Theoretical results are also often not helpful and the only final judgement must come from empirical tests. A well defined experimental methodology for doing this is helpful. We believe, and results in the International Timetabling Competition, support us, that experimental design and sequential testing are appropriate. The methodology that we adopted, based on the racing algorithm of Birattari et al. (2002) has the advantage of reducing the experiments to only those that are strictly necessary. As a matter of consequence, the number of configurations that is feasible to test in reasonable time increases. In our specific case the racing algorithm has been extended to the *"several runs on various instances"* experimental design, which is imposed when the number of available instances is small. Moreover, the race has been used in an interactive manner. This latter point is the main novelty of our methodological approach: the race, or sequential testing, is used not only for tuning SLS algorithms but mainly for guiding the design and development process of these algorithms by providing indications on which are the profitable heuristic rules and by inspiring new algorithms to be inserted in the race to compete against the previous survivors. By such refinements of well performing algorithms, it is possible to obtain in feasible times a final optimiser highly fitted for the application at hand.

We gave evidence that this methodology is successful. On the one side we showed that the algorithm developed in this manner ranked the best at the International Timetabling Competition. On the other side, a post development analysis unveiled the appropriateness of each algorithmic component and the correctness of their tuning. We showed that the construction heuristics can be simplified and that attaining feasible solutions is very easy and can be done quickly for the instances of the International Timetabling Competition. We then gave evidence of the importance of the Simulated Annealing part and investigated its profile which indicates the usefulness of the reheating feature. Finally, we considered qualified run time distributions and characterised the curves with exponential distributions thus unveiling that longer run time could be profitable for further improving solution quality and that apparently restarts of the algorithm are not needed.

Further analyses on the search landscape of the problem are contingent to the numerous simplifications that we applied. However evidence arose that the quality of local optima found after the Variable Neighbourhood Descent is correlated with the final solution quality. This fact makes it possible to predict the final quality of a solution after few seconds of running the algorithm, or, equivalently, to understand whether an instance is easy to solve or not. We were instead unable to find any correlation between distances of local optima and global optima in the search space while we substantiated with empirical observations the conjectured connectivity of the search landscape and the presence of large plateaux. In addition to landscape analysis, we attempted to derive a model for the prediction of the hardness of an instance from its *a priori* features. Enough evidence arose that the distribution of vertex degree in the constraint graph derived from the instance and the number of rooms and events are good predictors. Such kind of analysis may be particularly useful in a university that could then undertake corrective measures for improving its timetabling acting on the factors of main importance.

The work in this chapter advances the application of SLS methods in the specific area of timetabling. The main characteristic of timetabling is the high specificity of its constraints that vary from situation to situation. SLS methods are appealing in such a context because they are flexible and attain satisfiable performance. Nevertheless, it is desirable to reduce the efforts inherent in their application. Case-based systems (Burke et al., 2004b), hyper-heuristic systems (Burke et al., 2003), and frameworks for metaheuristics go in this direction. Our methodological approach based on methods from statistics is complementary to all these systems and above all it is the only one that can provide with an objective guarantee of performance, this latter aspect being of main importance in a context like timetabling where, due to the high specificity of the problem, the way to assess solvers is not yet fully understood.

Chapter 7.

Conclusions

In which we summarise the advances in the application of SLS methods which have been achieved with this thesis.

7.1. Main themes of this thesis

In this thesis, we put forward the view of SLS algorithms as modular methods with the main components being Construction Heuristics, Iterative Improvement procedures, and Metaheuristics. We maintained the analysis of these main components separated, and studied the computational results of their combinations. These combinations can be regarded as *instantiations* of general SLS methods for solving a specific problem. This way of proceeding implies a certain involvement in coding and the assembly of different components. We contributed to make this "SLS algorithm engineering" process more effective by defining a systematic methodology for its guidance based on empirical tests.

A central issue in the methodology is the selection among several alternative algorithmic configurations. For this goal, the definition of quantitative evaluation methods and assessment tools that take into account the stochastic nature of these algorithms is of essential importance. This should, in fact, be one of the priorities of scientific research in general, as emphasised in a recent report to the European Commission by a panel of experts (Bibel et al., 2004). In our field, the need for these tools is amplified by the recent explosion of publications on SLS algorithms all claiming the competitiveness of their new algorithms but, ultimately, leaving the SLS practitioner confused and overwhelmed with sometimes contradicting pieces of information.

We addressed the issue of defining quantitative methods for the comparison of algorithms for optimisation in Chapter 3, with particular emphasis on the Experimental Design theory from the field of Statistics. The defined methods are used for the comparison of SLS algorithms for two standard problems, the graph colouring problem and one of its most important generalisations, the set T-colouring problem. The results of our analysis are replicable and show which are the most appropriate SLS components and algorithms for these problems. The study is extensive and gives comprehensive results, similar in spirit to previous studies on the travelling salesman problem or the satisfiability problem in propositional logic. Systematic experimental analyses on standard problems are necessary in the field of SLS algorithms in order to create the basis for the selection of SLS components when new problems are faced and for the understanding of general guidelines on the application of these techniques.

We also applied statistical methods in the practical process of engineering SLS algorithms for solving a realistic timetabling problem. In this case, we could base the design of the SLS components on those developed for the two previous problems, given their relatively close relationship (graph colouring is at the core of timetabling problems). Their implementation and combination is conducted interactively with experiments and analysis of results. This case study exemplifies a methodology for the application of SLS methods that consists in three phases: (1) modelling the problem, (2) devising SLS components by recognising underlying standard problems, (3) adapting, configuring and combining these components for the problem at hand by interactive testing.

7.2. Contributions and results

The following is a detailed summary of the main contributions and results of this thesis.

On statistical methods for the analysis of SLS algorithms

We defined the experimental design scenarios that are appropriate for the analysis of stochastic optimisers like SLS algorithms. For each scenario, we indicated the appropriate tests statistics for three statistical approaches: parametric, rank-based, and permutation tests. The parametric analysis (ANOVA) entails complicated transformations of results to meet the assumptions of normality, independence, and homoschedasticity of the observed data. Despite these efforts, often these assumptions are not met and indeed their validity for results from stochastic optimisers may be arguable. Non-parametric methods are safer alternatives in this case. Rank-based tests are well known non-parametric methods with a large amount of research efforts and literature behind. Permutation tests are not yet well established and their use in the analysis of algorithms is new. In particular, we focused on *synchronised* permutation tests for two-way factorial designs, and for all-pairwise comparisons carried out by means of confidence intervals. In this latter context, the algorithms for the statistical analysis that we developed make a new contribution to the application of permutation tests.

A simulation study to compare the type I error rate and the statistical power of these three tests revealed that parametric tests are quite robust against the violation of the assumption of normality of distributions, while permutation tests require further efforts in their calibration. Rank-based tests result, however, to be comparable in power even with parametric tests under the cases considered and are, hence, preferable because of the fewer assumptions on the underlying data with respect to parametric tests.

Besides this, we extended a classical graphical representation of confidence intervals for the parametric case to the two other non-parametric tests. This representation allows for an immediate and easy evaluation of algorithms by visual inspection and is a helpful tool to support tables with numerical results. Most importantly, it allows to summarise a large number of data from statistical tests, *i.e.*, results on many instances and comparisons of many algorithms.

On the graph colouring

Experimental analysis. We carried out a systematic analysis of algorithms for solving the chromatic number problem. We clearly defined the experimental setting, implemented all the algorithms under study in a same framework, and allowed each algorithm to make the same use of computational resources. This guarantees the replicability of our analysis and emphasises the heuristic principles of the algorithms avoiding biases of implementation details. For the analysis of results we relied on statistical tests based on ranks. This study is an advancement with respect to previous analyses of stochastic optimisers for the graph colouring problem which were based on best results, frequency of attainment of best solutions, and widely different experimental settings.

Analysis of construction heuristics. We considered three known construction heuristics: the greedy sequential heuristic, DSATUR, and RLF. The superiority of RLF over the others with respect to solution quality was known. Our contribution has been to extend this result to several classes of instances, to support it by sound statistical evidence, and to provide an indication of the computational costs of all three heuristics.

A very large scale neighbourhood structure. We throughly studied the application of a very large scale neighbourhood for local search. In particular, we focused on a cyclic exchange neighbourhood structure. This neighbourhood is of exponential size but it can be examined effectively by solving an *all-pairs shortest path problem* in a corresponding *improvement graph*. We gave a dynamic programming algorithm for solving exactly this problem, that is inspired by the work of Dumitrescu (2002). We then extended this algorithm to return also improving path exchanges. We showed analytically the desirability of the union of these two neighbourhood structures that entails a search space with less local optima then smaller neighbourhoods like the one-exchange neighbourhood, and we quantified experimentally the incidence of this feature on the quality of solutions for small graphs. Yet, when larger graphs are to be solved, the neighbourhood examination algorithm becomes very costly and heuristic rules must be adopted for truncating it when opportune. We analysed these rules showing that, despite the loss of some cyclic and path exchanges, the new neighbourhood remains preferable to smaller ones.

Nevertheless, when the cyclic and path exchange neighbourhood is used together with metaheuristics like Tabu Search we registered an "unexpected failure", since a Tabu Search with a one-exchange neighbourhood performed better by quite a large margin. This shows that promising very large scale neighbourhoods may be difficult to exploit in practice. Only computational experiments may ascertain their competitiveness and it is, therefore, important to report also such negative results.

We indicated in the large loss of promising path exchanges the main cause of this failure. This loss is mainly due to the heuristic rules adopted. Unfortunately, devising similar rules that guarantee path exchanges to be maintained and, at the same time, provide significant reduction of the search effort, emerged to be infeasible within our algorithmic framework.

Comparison of SLS algorithms. We re-implemented and studied five of the best known and best performing approximate algorithms for graph colouring: Tabu Search with one-exchange moves (originally by Hertz and de Werra, 1987, and later improved by Dorne and Hao, 1998a), a Tabu Search variant of the Min-Conflicts heuristic (Stützle, 1998), the Hybrid Evolutionary Algorithm (Galinier and Hao, 1999), Simulated Annealing with

Kempe chains, and XRLF (these last two both by Johnson et al. 1991). For this latter algorithm, we studied in-depth the behaviour of its components in order to include an automatic tuning of some of its parameters for the final comparison.

In addition to Tabu Search with a very large scale neighbourhood, we applied for the first time other SLS methods to the graph colouring problem. These are Iterated Local Search, Guided Local Search and Novelty$^+$ which were chosen because of their good performance on related problems such as the maximum satisfiability problem and constraint satisfaction problem. The main results of the study are the following:

- The simple Tabu Search with one-exchange neighbourhood remains a very competitive algorithm and often its combination with further Metaheuristics does not introduce any significant further improvement. It gives the best run-time versus solution quality trade off.
- Besides Tabu Search, a new result is the superiority of Guided Local Search on Geometric graphs or, more in general, on graphs whose clique number is very close to the chromatic number.
- The performance of the Hybrid Evolutionary Algorithm resulted to be inferior to what was expected. Its results are significantly the best only on one class of instances. Even favourable results on random graphs with edge density 0.5 are not confirmed by our large size experiments. We pointed out that the reason for such differences may be the relatively short running-time allowed in this work and that HEA requires an instance specific parameter tuning.
- Other two well known algorithms, Simulated Annealing and XRLF, are only rarely competitive. Given also the difficulty of tuning these methods they are not recommendable.

Problem instances. We observed that graphs coming from real life applications are easier to solve than random graphs. The study of random graphs remains nevertheless meaningful because algorithms that perform well on challenging graphs perform well also on real world graphs. We used an additional, new set of 1260 random graphs to make the results of the comparison among algorithms more reliable. Graphs that have a clique number very close to the chromatic number are easily solvable by Ex-DSATUR implemented by Mehrotra and Trick (1996) even if their size is 1000 vertices. Geometric random graphs often exhibit this property. However, if Ex-DSATUR does not finish in short time it will probably require exponentially longer computation times and in that case SLS algorithms provide much better results than simple construction heuristics or prematurely stopped exact algorithms. A Guided Local Search algorithm with one-exchange resulted as the most suitable algorithm for solving geometric graphs. This algorithm has been newly developed for the first time and constitutes an original contribution of this thesis.

On the graph colouring generalisations: the set T-colouring problem

The set T-colouring problem corresponds to the frequency assignment problem when the latter is polished from specific technicalities; therefore, we extended our study to algorithms proposed in that context. We focus on the common objective of minimising

the span of the colouring, that is, the difference between the largest and the smallest colour used.

Construction heuristics. Many possible choices arise for assembling construction heuristics. We presented an extensive study to identify the best combination of choices by using a factorial design. The previous belief was that the available heuristics rank differently on the instances and that the best possible approach was to try them all. We showed instead that, for both Uniform and Geometric random graphs with a defined group of constraints, this is not true, and that a generalised form of DSATUR is clearly superior to all other heuristics. This heuristic works on the T-colouring transformation of the set T-colouring problem.

Comparison of SLS algorithms. The known SLS algorithms included in the analysis were a Tabu Search with one-exchange neighbourhood (Dorne and Hao, 1998b; Hao et al., 1998), an incomplete randomised Backtracking algorithm (Prestwich, 2002b), and an extension of the Adaptive Iterated Greedy algorithm combined with Tabu Search (Lim et al., 2003). The re-implementation of this algorithm enhanced its performance with respect to the original proposal that left several implementation details open (Lim et al., 2003). In addition, we adapted the most promising algorithms for the "classical" graph colouring problem, such as the Hybrid Evolutionary algorithm, Guided Local Search, and the Min-Conflicts heuristic that were all used for solving the set T-colouring problem transformed into the T-colouring problem. Finally, we enhanced the one-exchange neighbourhood with the possibility to reassign all the colours of a single vertex in such a way that all the constraints involving that vertex are satisfied. The main results of the comparison are the following:

- Among the different approaches, the best is solving a sequence of set T-colouring decision problems. Moreover, the local search methods studied work better on the original problem rather than on its transformation in T-colouring problem. The former approach implies the use of the vertex constraints in the definition of the solution representation, thus, restricting the search space.
- On Geometric random graphs, the best performing algorithm is clearly Adaptive Iterated Greedy and the results presented improve the best known results on 25 out of 28 instances.
- On Uniform random graphs, the best overall algorithm is still Adaptive Iterated Greedy but on graphs of edge density around 0.5 it is outperformed by a Tabu Search with the one-exchange neighbourhood enlarged by exact reassignments of colours at the vertices. These results remain robust to a certain degree of variability of the vertex requirements and distance constraints.

We gave evidence that solving the set T-colouring problem is much more computationally expensive than solving the classical graph colouring problem. Two consequences derive from this fact. On the one side, exact algorithms are impracticable even on graphs that can be easily solved when used them as graph colouring problem instances. On the other side, SLS algorithms require much longer run time and even for the longest run times attempted, the probability of finding improvements remained high. This compelled us to reduce considerably the size of the instances used in the experiments with respect to those for the classical graph colouring.

On the timetabling

We have been concerned with a particular case of university course timetabling. This problem has been proposed in the context of the International Timetabling Competition and included some of the constraints that often arise in practical cases. More than one thousand algorithmic configurations were tested, including several Construction Heuristics, several neighbourhood structures for Iterative Improvement, and several Metaheuristics. The selected algorithm resulted to be the best at the International Timetabling Competition. It is based on a framework that may be reusable. It first finds a feasible solution by satisfying the hard constraints and then improves this feasible solution by minimising the violations of soft constraints. It applies, sequentially, Construction Heuristics, and Iterative Improvement algorithms in several neighbourhoods in the fashion of Variable Neighbourhood Descent. For finding feasible solutions with respect to the hard constraints, it uses Tabu Search. Finally, when all previous methods have lost their capability to provide improvements, it continues with Simulated Annealing, which gives its contribution in the long term by slowly improving the solution quality in a large neighbourhood. A further feature of the algorithm is the exact reassignment of rooms in some phases of the search.

Such a complex algorithm could be assembled and tuned only thanks to the use of a systematic methodology for algorithm selection. This methodology is based on the interaction between development and experimental analysis which is carried out by sequential testing like in the racing algorithm of Birattari (2004b). The success in the competition sheds credit on the appropriateness of such a methodology in the application of SLS algorithms whose behaviour is hard to forecast and can only be ascertained by computational experiments.

Our work makes also a contribution to the specific field of timetabling indicating for the existing algorithmic frameworks in this field the appropriate tools for the automatic or semi-automatic selection of configurations. This is chiefly important in a field where the specification of the problem is almost always different from case to case.

7.3. Open issues

It is now a few decades that academic research has been studying Stochastic Local Search techniques. They became well established methods for solving large scale optimisation problems and are widespread in industry. Yet, very little is understood about these techniques and how algorithm components can be matched to problem characteristics.

We believe that the most important advance for the application of SLS techniques in the next years should be the development and refinement of a theory of practice. Prescriptive rules may guide only an high level configuration of the algorithms and have only weak impact. An experimental approach for the final configuration of these algorithms is needed and we showed in this thesis which are the methods for this approach and how they can be applied in practice. Yet, the elements in use in our proposed "SLS engineering approach" still require further progress. We identify three different levels for future research. These levels act on different degrees of detail and on different issues concerned with SLS techniques. To a certain extent, they all have been touched in this thesis but a lot more needs to be done because of the complexity of the field.

7.3.1. Experimental methodology

In this context, the practice of SLS techniques may borrow methods from biostatistics. Differently from biology and medicine, optimisation problems are abstract and can be studied mathematically. Nevertheless, we showed that the impact of theoretical results must be verified in practice and even exact algorithms must be tested empirically because their behaviour in terms of computation time depends strongly on the particular problem instance to be solved. There is the need to use empirical tests to classify the algorithms on different combinatorial problems and on various instance classes that have relevant practical applications. This situation is similar to the field of biostatistics, where treatments are to be classified for different pathologies and on different typologies of patients. In that field design and analysis of computational experiments are used to extract reliable knowledge from clinical trials. Similarly, these methodologies are helpful to organise the knowledge in the field of combinatorial optimisation. This approach has not yet been followed in a systematic manner and in some contexts of analysis the definition of the methods for a correct design and test of experiments is still missing. In this work in Chapter 3, we defined the experimental methodology for testing SLS algorithms when their performance is assessed only by the quality of their approximation to the optimal solution. A further development in this context might be the use of tests that do not rely on the assumption of homoschedasticity of results. Nevertheless, the univariate case of solution quality is only one of the possible relevant scenarios in the analysis of algorithms for optimisation. The definition of the correct methods of analysis is urgent in the following further contexts.

- assessment of algorithm performance through multivariate measures, such as the combination of solution quality and computation time for its attainment;
- comparison of algorithms' profile curves that describe the development of solution quality over time;
- classification of algorithms matching instance features.

7.3.2. A library of basic SLS components

A central goal of academic studies is mapping SLS components with best performance on standard problems, like the graph colouring problem and its generalisations. The classification should be enough fine-grained in order to distinguish among instance characteristics, solution quality and running time. It should also provide the best set of parameters required by the algorithms. The statistical tools envisaged in the previous point should be used for accomplishing this classification.

Ideally, this collection should be supported by publically available libraries with highly optimised modules for accomplishing well specified tasks. For example, optimised neighbourhood search is a very important element of SLS algorithms and our experience showed that the use of several neighbourhoods in a Variable Neighbourhood fashion is often a profitable choice. Such libraries would be essential to speed up the work of SLS practitioners who are typically overwhelmed by other technicalities when facing real life applications.

The SLS practitioner could recognise the type of problem, consult the library, and select the components that best match its application scenario, that is, with the features of the instances he has to solve in a given time. In a next step, he can include or adapt the pieces of code for its own more complex application. Using an hazardous parallel with medicine, this process resembles the process of a doctor while serving medical care. He visits his patient and recognises some symptoms typical of some well-known diseases. Then, he consults a database where diseases and best cures are archived and, finally, prescribes one or more opportune treatment. The archive is organised according to the results given by studies of bio-statisticians. The SLS practitioner act like the doctor and our prefigured collection of SLS components for standard problems takes the place of the archive. Clearly, the archive may be extended by the experience and additional pieces of information of the SLS practitioner but making knowledge transmissible is a higher achievement that must be pursued.

7.3.3. The algorithmic context

The central issue in the field of SLS methods for combinatorial optimisation remains however the improvement of the existing algorithms. Construction heuristics and neighbourhood structures are very important and often determine the success of SLS algorithms. New ideas in these and other SLS components are possible and require detailed studies on specific combinatorial optimisation problems, similar to the one presented here for the graph colouring problem and the set T-colouring problem.

An important issue that needs further investigation is the understanding of the cases where VLSNs can be effective. We presented a negative example on the graph colouring problem but for the travelling salesman problem a large neighbourhood is at the core of the best performing SLS algorithms. Moreover, even on the graph colouring problem, the attempt to use a VLSN proposed in this work should not be considered exhaustive. In particular, the empirical results presented in this thesis indicated that path exchanges are more helpful than cyclic exchanges. It could then be worth trying to devise a different algorithm that searches for path exchanges and omitts cyclic exchanges. In addition, it is certainly necessary to enlarge the application of the cyclic exchange neighbourhoods to classes of graphs with different structure with respect to the random graphs. Indeed, the graph structure is very likely to influence considerably the performance of algorithms and there may exist special graphs in which the contribution of the cyclic exchange neighbourhood becomes prominent.

Further developments of SLS algorithms for graph colouring are possible even at a more general level, as many ideas remained unattempted. Above all, we believe that the potential of Iterated Local Search has not yet been fully exploited. In recent publications (Williams et al., 2003; Glass and Prügel-Bennett, 2005), ideas arose for profitably modifying or precolouring part of the graph that captures the overall combinatorics of a problem instance. It would be interesting, then, to study similar ideas for obtaining perturbations in Iterated Local Search.

In addition to this, the generalisations of graph colouring gives rise to a class of problems that is particulary interesting for its many real world applications but that has not yet been studied comprehensively. These problems exhibit different types of constraints whose influence would be worthwhile to be analysed. The generalisation of existing heuristics for solving the "classical" graph colouring problem is not guaranteed to be the best possible choice and specialised heuristics can be devised for each case. Hence, these

problems are the ideal context for further developments of SLS techniques on assignment problems.

In conclusion, the literature on SLS methods has shown that large scale experimental studies like those for the travelling salesman problem and satisfiability problem in propositional logic lead to a significant progress in the research on combinatorial optimisation. We hope that the new studies presented here and the new methodological techniques applied make a very significant contribution to this type of research and that they further foster interest in a theory of practice of SLS algorithms.

Appendix A.

A Formal Study on Iterative Improvement for Graph Colouring

In this appendix, we present an analytical study of the neighbourhood structures for local search in graph colouring. More specifically, we focus on the iterative improvement paradigm with best improvement strategy for solving the k-colouring problem. The solution representation is a complete (possibly infeasible) colouring and the objective function to minimise is the number of edges that are in conflict in the colouring. For an introduction to the notation and the definition of the neighbourhood structures, we refer the reader to the Sections 4.2 and 4.8.1, respectively.

The appendix is organised in two sections. In the first, we formalise the conditions for local optimality for each neighbourhood. In doing this, we also show that the evaluation of neighbours can be done in constant time by maintaining some auxiliary data structures during the search. Updating these auxiliary data structures at each step of the local search can be accomplished in $O(|V|)$.

In the second section, we give the formal proof of the claim that the use of cyclic and path exchanges in the neighbourhood structure increases the number of infeasible colourings in the search space that are not better than those in which the local search ends. This effect can be formalised through the concept of *dominance number* introduced by Glover and Punnen (1997) (see also Gutin et al. 2003, 2005). This corresponds to the first attempt to show analytically the superiority of a local search method through dominance analysis in the graph colouring problem.

A.1. The conditions of local optimality

We pose here the conditions of local optimality in each neighbourhood structure. We first introduce the following propositions and remarks that follow from the definitions given in Section 4.2:

Remark A.1 *The particular case* $|A_{\{u\}}(v)| = |A_{\{v\}}(u)|$*, with* $u, v \in V$*, is* 1 *if* $(u, v) \in E$ *and* 0 *otherwise.*

Proposition A.1 *Given two sets of vertices* T *and* U*, both subsets of* V*, the number of edges connecting vertices in* T *with vertices in* U *is the sum of the number of vertices in* $U \setminus T$ *adjacent to vertices in* T *and half of the number of vertices in* T *adjacent to vertices in* T*,* i.e.*,*

$$|E_U(T)| = \sum_{v \in T} |A_{U \setminus T}(v)| + \frac{1}{2} \sum_{v \in T} |A_T(v)|, \forall T \subseteq U \subseteq V.$$

Proof. The Proposition is an immediate consequence of Remarks 4.1 and 4.2.

$$\begin{aligned}
|E_U(T)| &= \Big|\bigcup_{v\in T} E_U(v)\Big| = \Big|\bigcup_{v\in T} E_{U\setminus T}(v) \cup \bigcup_{v\in T} E_T(v)\Big| = \\
&= \Big|\bigcup_{v\in T} E_{U\setminus T}(v)\Big| + \Big|\bigcup_{v\in T} E_T(v)\Big| = \qquad \text{(for Remark 4.2)} \\
&= \sum_{v\in T} |E_{U\setminus T}(v)| + \frac{1}{2}\sum_{v\in T} |E_T(v)| = \qquad \text{(for Remark 4.1)} \\
&= \sum_{v\in T} |A_{U\setminus T}(v)| + \frac{1}{2}\sum_{v\in T} |A_T(v)|.
\end{aligned}$$

□

Proposition A.2 *Given two sets of vertices T and U, $T \subseteq U$, both subsets of V, and a set $Z : Z \cap U = \emptyset$, (hence, also $Z \cap T = \emptyset$) it is:*

$$\begin{aligned}
|E_{U\setminus T\cup Z}(|) &= \frac{1}{2}\sum_{x\in U} |A_U(x)| + \frac{1}{2}\sum_{x\in T} |A_T(x)| + \frac{1}{2}\sum_{x\in Z} |A_Z(x)| + \\
&\quad + \sum_{x\in Z} |A_U(x)| - \sum_{x\in Z} |A_T(x)| - \sum_{x\in T} |A_U(x)|
\end{aligned}$$

Proof. From Remarks 4.2 and 4.1 it derives that

$$|E_{U\setminus T\cup Z}(|) = \frac{1}{2}\sum_{x\in U} |A_{U\setminus T\cup Z}(x)| - \frac{1}{2}\sum_{x\in T} |A_{U\setminus T\cup Z}(x)| + \frac{1}{2}\sum_{x\in Z} |A_{U\setminus T\cup Z}(x)|.$$

For the hypotheses on T, U and Z, each term of the sum can be decomposed in:

$$\begin{aligned}
\sum_{x\in U} |A_{U\setminus T\cup Z}(x)| &= \sum_{x\in U} |A_U(x) \setminus A_T(x) \cup A_Z(x)| = \\
&= \sum_{x\in U} |A_U(x)| - \sum_{x\in U} |A_T(x)| + \sum_{x\in U} |A_Z(x)|, \\
\sum_{x\in T} |A_{U\setminus T\cup Z}(x)| &= \sum_{x\in T} |A_U(x)| - \sum_{x\in T} |A_T(x)| + \sum_{x\in T} |A_Z(x)|, \\
\sum_{x\in Z} |A_{U\setminus T\cup Z}(x)| &= \sum_{x\in Z} |A_U(x)| - \sum_{x\in Z} |A_T(x)| + \sum_{x\in Z} |A_Z(x)|,
\end{aligned}$$

and the rest of the claim follows trivially by noting that

$$\sum_{t\in T} |A_U(t)| = \sum_{t\in T}\sum_{u\in U} |A_u(t)| = \sum_{u\in U}\sum_{t\in T} |A_t(u)| = \sum_{u\in U} |A_T(u)|$$

and substituting the terms in the sum. □

The following remarks descend from Proposition A.2.

Remark A.2 $|E_{U\setminus\{u\}}(|) = \frac{1}{2}\sum_{x\in U} |A_U(x)| - |A_U(u)|, \quad \forall U \subseteq V, u \in U.$

Remark A.3 $|E_{U\cup\{v\}}(|) = \frac{1}{2}\sum_{x\in U} |A_U(x)| + |A_U(v)|, \quad \forall U \subseteq V, v \in V \setminus U.$

Remark A.4 $|E_{U\setminus\{u\}\cup\{v\}}(|) = \frac{1}{2}\sum_{x\in U}|A_U(x)| + |A_U(v)| - |A_U(u)| - |A_{\{u\}}(v)|, \quad \forall U \subseteq V, u \in U, v \in V \setminus U$.

We have now the elements to express analytically the conditions of local optimality of an infeasible colouring $\mathcal{C}$ with respect to a given neighbourhood structure.

Theorem A.1 *A k-colouring $\mathcal{C}$ is a local optimum for the neighbourhood structure $\mathcal{N}_1$ if and only if $\forall v \in V^c$ and $\forall i \in \Gamma \setminus \{\varphi(v)\}$:*

$$|A_{C_{\varphi(v)}}(v)| \leq |A_{C_i}(v)|$$

Proof. In $\mathcal{N}_1$ a neighbour $\mathcal{C}'$ is obtained from $\mathcal{C}$ by replacing $C_{\varphi(v)}$ with $C'_{\varphi(v)} = C_{\varphi(v)} \setminus \{v\}$ and C_i with $C'_i = C_i \cup \{v\}$ for a vertex $v \in V$ and a colour $i \in \Gamma$. Hence,

$$f(\mathcal{C}') \quad = \quad f(\mathcal{C}) - |E^c_{C_{\varphi(v)}}| + |E^c_{C'_{\varphi(v)}}| - |E^c_{C_i}| + |E^c_{C'_i}|$$

Using Remark A.2 to make explicit the terms and simplifying, it follows that:

$$f(\mathcal{C}') = f(\mathcal{C}) - |A_{C_{\varphi(v)}}(v)| + |A_{c_i}(v)|$$

which, according to the definition of local optimum (Definition 2.1), proves the statement. □

Theorem A.2 *A k-colouring $\mathcal{C}$ is a local optimum for the neighbourhood structure $\mathcal{N}_2$ if and only if $\forall v \in V^c$ and $\forall u \in V$:*

$$|A_{C_{\varphi(v)}}(v)| + |A_{C_{\varphi(u)}}(u)| \leq |A_{C_{\varphi(u)}}(v)| + |A_{C_{\varphi(v)}}(u)| - 2 \cdot |A_{\{u\}}(v)|$$

Proof. In $\mathcal{N}_2$ a neighbour $\mathcal{C}'$ is obtained from $\mathcal{C}$ by replacing $C_{\varphi(u)}$ and $C_{\varphi(v)}$ respectively with $C'_{\varphi(u)} = C_{\varphi(u)} \setminus \{u\} \cup \{v\}$ and $C'_{\varphi(v)} = C_{\varphi(v)} \setminus \{v\} \cup \{u\}$ for a pair of vertices $u, v \in V$. Hence,

$$f(\mathcal{C}') = f(\mathcal{C}) - |E^c_{C_{\varphi(u)}}| + |E^c_{C'_{\varphi(u)}}| - |E^c_{C_{\varphi(v)}}| + |E^c_{C'_{\varphi(v)}}|$$

Using Remark A.4 and simplifying it follows that:

$$f(\mathcal{C}') = f(\mathcal{C}) - |A_{C_{\varphi(v)}}(v)| + |A_{C_{\varphi(v)}}(u)| - |A_{C_{\varphi(u)}}(u)| + |A_{C_{\varphi(u)}}(v)| - 2 \cdot |A_{\{u\}}(v)|$$

from which the statement descends trivially. □

Theorem A.3 *A k-colouring $\mathcal{C}$ is a local optimum for the neighbourhood structure $\mathcal{N}_3$ if and only if $\forall u, v \in V : \varphi(u) = \varphi(v), (u, v) \in E$, and $\forall w \in A_V(u), x \in A_V(v) : \varphi(w) \neq \varphi(x)$:*

$$\begin{aligned}|A_{C_{\varphi(u)}}(u)| + |A_{C_{\varphi(v)}}(v)| + |A_{C_{\varphi(w)}}(w)| + |A_{C_{\varphi(x)}}(x)| \leq \\ |A_{C_{\varphi(w)}}(u)| + |A_{C_{\varphi(x)}}(v)| + |A_{C_{\varphi(u)}}(w)| + |A_{C_{\varphi(u)}}(x)| - 3 - |A_{\{v\}}(w)| - |A_{\{u\}}(x)|\end{aligned}$$

Proof. In $\mathcal{N}_3$ a neighbour $\mathcal{C}'$ is obtained from $\mathcal{C}$ by replacing $C_{\varphi(u)}$, $C_{\varphi(w)}$ and $C_{\varphi(x)}$ respectively with $C'_{\varphi(u)} = C_{\varphi(u)} \setminus \{u, v\} \cup \{w, x\}$, $C'_{\varphi(w)} = C_{\varphi(w)} \setminus \{w\} \cup \{u\}$ and $C'_{\varphi(x)} = C_{\varphi(x)} \setminus \{x\} \cup \{v\}$ for the vertices $u, v \in V^c$, $w \in A_V(u)$, and $x \in A_V(v)$. Hence,

$$f(\mathcal{C}') = f(\mathcal{C}) - |E^c_{C_{\varphi(u)}}| + |E^c_{C'_{\varphi(u)}}| + |E^c_{C_{\varphi(w)}}| + |E^c_{C'_{\varphi(w)}}| + |E^c_{C_{\varphi(x)}}| + |E^c_{C'_{\varphi(x)}}|$$

Using Proposition A.2 on $|E^c_{C'_{\varphi(u)}}|$ and Remark A.4 on the other terms, it follows that:

$$\begin{aligned} f(\mathcal{C}') &= f(\mathcal{C}) + \sum_{t\in\{w,x\}} |A_{C_{\varphi(u)}}(t)| - \sum_{t\in\{w,x\}} |A_{\{u,v\}}(t)| - \sum_{t\in\{u,v\}} |A_{C_{\varphi(u)}}(t)| + \\ &+\frac{1}{2}\sum_{t\in\{u,v\}} |A_{\{u,v\}}(t)| \overset{1}{+} \frac{1}{2}\sum_{t\in\{w,x\}} |A_{\{w,x\}}(t)| \overset{0}{+} \\ &-|A_{C_{\varphi(w)}}(w)| + |A_{C_{\varphi(w)}}(u)| - |A_{\{w\}}(u)| - |A_{C_{\varphi(x)}}(x)| + |A_{C_{\varphi(x)}}(v)| - |A_{\{x\}}(v)| \end{aligned}$$

and substituting $|A_{\{u\}}(v)| = |A_{\{v\}}(u)| = |A_{\{w\}}(u)| = |A_{\{x\}}(v)| = 1$ as from the definition of the neighbourhood (Definition 4.3):

$$\begin{aligned} f(\mathcal{C}') = f(\mathcal{C}) - |A_{C_{\varphi(u)}}(u)| - |A_{C_{\varphi(v)}}(v)| - |A_{C_{\varphi(w)}}(w)| - |A_{C_{\varphi(x)}}(x)| + |A_{C_{\varphi(u)}}(w)| + \\ + |A_{C_{\varphi(u)}}(x)| + |A_{C_{\varphi(w)}}(u)| + |A_{C_{\varphi(x)}}(v)| - 3 - |A_{\{v\}}(w)| - |A_{\{u\}}(x)| \end{aligned}$$

that, together with the definition of local optimum (Definition 2.1), proves the statement. □

Theorem A.4 *A k-colouring $\mathcal{C}$ is a local optimum for the neighbourhood structure $\mathcal{N}_4$ if and only if $\forall\, m \leq k$ and $\forall\, V' = \{v_1, \ldots, v_m\} \subseteq V : \varphi(v_i) \neq \varphi(v_j), \forall\, i \in \{1, \ldots, m\}$:*

$$\sum_{i=1}^{m} |A_{C_{\varphi(v_i)}}(v_i)| \leq \sum_{i=2}^{m} \Big(|A_{C_{\varphi(v_i)}}(v_{i-1}) - |A_{v_{i-1}}(v_i)| \Big) + |A_{C_{\varphi(v_1)}}(v_m)| - |A_{v_1}(v_m)|$$

Proof. We consider a colouring $\mathcal{C}'$ obtained by $\mathcal{C}$ by a cyclic exchange of the set of vertices $\{v_1, \ldots, v_m\}$ as defined by Definition 4.4. We denote $C'_{\varphi(v_i)} = C_{\varphi(v_i)} \setminus \{v_i\} \cup \{v_{i-1}\}, \forall i \in \{1, \ldots, m\}$, with the notational artifice of $v_0 \equiv v_m$. Then it is

$$f(\mathcal{C}') = f(\mathcal{C}) - \sum_{i=1}^{m} |E^c_{C_{\varphi(v_i)}}| + \sum_{i=1}^{m} |E^c_{C'_{\varphi(v_i)}}|$$

Using Remark A.4 and simplifying we obtain

$$f(\mathcal{C}') = f(\mathcal{C}) + \sum_{i=1}^{m} \Big(|A_{C_{\varphi(v_i)}}(v_{i-1})| - |A_{C_{\varphi(v_i)}}(v_i)| - |A_{v_{i-1}}(v_i)| \Big)$$

from which, rewriting the sum avoiding the use of v_0, the statement is derived. □

Theorem A.5 *A k-colouring $\mathcal{C}$ is a local optimum in the neighbourhood structure $\mathcal{N}_5$ if and only if $\forall m \leq k$ and $\forall\, V' = \{v_1, \ldots, v_{m-1}\} \subseteq V : \varphi(v_i) \neq \varphi(v_j), \forall i \in \{1, \cdots, m-1\}$:*

$$\sum_{i=1}^{m-1} |A_{C_{\varphi(v_i)}}(v_i)| \leq \sum_{i=1}^{m-1} |A_{C_{\varphi(v_{i+1})}}(v_i)| - \sum_{i=1}^{m-2} |A_{\{v_i\}}(v_{i+1})|$$

Proof. Similarly to Theorem A.4 we prove also this statement by induction. The case $m = 2$ corresponds to Theorem A.1 and it has been already proved. We show that, if the statement holds for a neighbour $\mathcal{C}'$ obtained by replacing in $\mathcal{C}$ the colour classes containing the vertices $V' = \{v_1, v_2, \ldots, v_m\} \subset V$ by a path exchange of V' as from

Definition 4.5, it holds also for a neighbour $\mathcal{C}''$ obtained from $\mathcal{C}$ by replacing the colour classes containing the vertices $V'' = V' \cup \{v_m\} \subseteq V$ by a path exchange of V'' as from Definition 4.5 .
Indeed, it is $C'_{\varphi(v_i)} = C''_{\varphi(v_i)} = C_{\varphi(v_i)} \setminus \{v_i\} \cup \{v_{i-1}\}, \forall i \in \{1, \ldots, m-2\}$, $C'_{\varphi(v_{m-1})} = C_{\varphi(v_{m-1})} \cup \{v_{m-2}\}$, $C''_{\varphi(v_{m-1})} = C_{\varphi(v_{m-1})} \setminus \{v_{m-1}\} \cup \{v_{m-2}\}$ and $C''_{\varphi(v_m)} = C_{\varphi(v_m)} \cup \{v_{m-1}\}$ and

$$
\begin{aligned}
f(\mathcal{C}'') &= f(\mathcal{C}) - \sum_{i=1}^{m} |E^c_{C_{\varphi(v_i)}}| + \sum_{i=1}^{m} |E^c_{C''_{\varphi(v_i)}}| = \\
&= f(\mathcal{C}) - \sum_{i=1}^{m-1} |E^c_{C_{\varphi(v_i)}}| + \sum_{i=1}^{m-1} |E^c_{C''_{\varphi(v_i)}}| - |E^c_{C_{\varphi(v_m)}}| + |E^c_{C''_{\varphi(v_m)}}| = \\
&= f(\mathcal{C}) - \sum_{i=1}^{m-1} |E^c_{C_{\varphi(v_i)}}| + \sum_{i=1}^{m-2} |E^c_{C'_{\varphi(v_i)}}| + |E^c_{C''_{\varphi(v_{m-1})}}| - |E^c_{C_{\varphi(v_m)}}| + |E^c_{C''_{\varphi(v_m)}}| = \\
&= f(\mathcal{C}) - \sum_{i=1}^{m-1} |E^c_{C_{\varphi(v_i)}}| + \sum_{i=1}^{m-1} |E^c_{C'_{\varphi(v_i)}}| + \\
&\quad + |E^c_{C''_{\varphi(v_{m-1})}}| - |E^c_{C'_{\varphi(v_{m-1})}}| - |E^c_{C_{\varphi(v_m)}}| + |E^c_{C''_{\varphi(v_m)}}|
\end{aligned}
$$

The first three terms of the sum correspond to $f(\mathcal{C}')$ whose value is known for hypothesis. The development of the other terms derives from Remarks A.3 and A.4:

$$
\begin{aligned}
f(\mathcal{C}'') &= f(\mathcal{C}) - \overbrace{\sum_{i=1}^{m-2} |A_{C_{\varphi(v_i)}}(v_i)|}^{A} + \overbrace{\sum_{i=1}^{m-2} |A_{C_{\varphi(v_{i+1})}}(v_i)|}^{B} - \overbrace{\sum_{i=1}^{m-3} |A_{\{v_i\}}(v_{i+1})|}^{C} + \\
&\quad + |A_{C_{\varphi(v_{m-1})}}(v_{m-2})| - |A_{C_{\varphi(v_{m-1})}}(v_{m-1})| - |A_{\{v_{m-2}\}}(v_{m-1})| \\
&\quad - |A_{C_{\varphi(v_{m-1})}}(v_{m-2})| + |A_{C_{\varphi(v_m)}}(v_{m-1})|
\end{aligned}
$$

from which, simplifying and grouping as indicated, the statement is derived. □

These theorems provide the exact formulas for evaluating neighbouring solutions in each respective neighbourhood structure by computing only the local changes. Local changes can be obtained simply by computing the values $|A_{C_i}(v_j)|$. Maintaining these values updated in an auxiliary matrix the evaluation of a neighbour can be done in $\mathcal{O}(1)$.

Beside this observation, the theorems introduced in this Section serve in the dominance analysis that follows.

A.2. Dominance relations between local searches

We analyse analytically the performance of an iterative improvement algorithm with best improvement strategy on the k-colouring problem. Such algorithms end in solutions that are local optima with respect to a certain neighbourhood structure and a given evaluation

function. We remark that the extension of a neighbourhood structure obtained by the union of two or more basic neighbourhoods does not necessarily imply an improvement in the capability of solving the problem. In the TSP, for example, it has been shown that a polynomial-time heuristic dominates exponentially large neighbourhoods (see Gutin et al. 2002 and Gutin et al. 2003). In a similar vein, we argue here that, in the case of the k-colouring problem, the extension of a neighbourhood is worth only if it makes possible to reach solutions of better quality. We focus therefore on the local optima produced by different iterative improvement algorithms and compare their quality. To help us in the discussion we conform to the notation used in *dominance analysis* (Gutin et al., 2005).

Let $\mathcal{A}$ be a heuristic algorithm and let $\mathcal{C}^{\mathcal{A}}(G,k)$ be the set of solutions obtained by $\mathcal{A}$ when solving the k-colouring problem with k colours on the graph G.

Definition A.1 The *domination number*, $domn(\mathcal{A},G,k)$, of an algorithm $\mathcal{A}$ on a graph G and k colours is the number of k-colourings which are not better than the solution $\mathcal{C}$ found by $\mathcal{A}$ (including $\mathcal{C}$ itself). That is, $domn(\mathcal{A},G,k) = |\{\mathcal{C} \in \mathcal{S}(G,k) : f(\mathcal{C}^{\mathcal{A}}(G,k)) \leq f(\mathcal{C})\}|$.

Definition A.2 We say that an algorithm $\mathcal{A}$ dominates an algorithm $\mathcal{B}$ if $domn(\mathcal{A},G,k) \leq domn(\mathcal{B},G,k)$ for all graphs G and integer k and there exists at least one graph G and integer k for which the inequality is strict. We denote this relation with $\mathcal{A} \succ \mathcal{B}$.

Clearly, if $domn(\mathcal{A},G,k) \leq domn(\mathcal{B},G,k)$ then it is $|\{\mathcal{C} \in \mathcal{S}(G,k) : f(\mathcal{C}^{\mathcal{A}}(G,k)) \leq f(\mathcal{C}(G,k))\}| \leq |\{\mathcal{C} \in \mathcal{S}(G,k) : f(\mathcal{C}^{\mathcal{B}}(G,k)) \leq f(\mathcal{C}(G,k))\}|$ which is certainly true in the case $\mathcal{C}^{\mathcal{A}}(G,k) \subseteq \mathcal{C}^{\mathcal{B}}(G,k)$. In order to show the dominance relation we need to show, then, that there exists at least one graph with one k such that $\mathcal{C}^{\mathcal{A}}(G,k) \subset \mathcal{C}^{\mathcal{B}}(G,k)$.

We consider the following iterative improvement algorithms with best improvement strategy obtained by combinations of neighbourhoods which are all extensions of the classical one exchange neighbourhood:

- Iterative_Improvement$_{\mathcal{N}_1 \cup \mathcal{N}_2}$,
- Iterative_Improvement$_{\mathcal{N}_1 \cup \mathcal{N}_2 \cup \mathcal{N}_3}$,
- Iterative_Improvement$_{\mathcal{N}_1 \cup \mathcal{N}_4}$,
- Iterative_Improvement$_{\mathcal{N}_2 \cup \mathcal{N}_5}$, and
- Iterative_Improvement$_{\mathcal{N}_4 \cup \mathcal{N}_5}$;

and we are interested in the dominance relation between these algorithms.

Theorem A.6 Iterative_Improvement$_{\mathcal{N}_1 \cup \mathcal{N}_2} \succ$ Iterative_Improvement$_{\mathcal{N}_1}$.

Proof. To prove the relation we have to show that:

(i) all local optima for the neighbourhood structure $\mathcal{N}_1 \cup \mathcal{N}_2$ are also local optima for the neighbourhood structure $\mathcal{N}_1$, and that

(ii) there exists an infeasible candidate solution which is local optimum in the neighbourhood structure $\mathcal{N}_1$ but not in the neighbourhood structure $\mathcal{N}_1 \cup \mathcal{N}_2$.

Since for each $\mathcal{C}$ the neighbourhood $\mathcal{N}_1(\mathcal{C}) \cup \mathcal{N}_2(\mathcal{C})$ contains all the candidate solutions of $\mathcal{N}_1(\mathcal{C})$, (i) is verified by definition.

Showing (ii) corresponds to finding a graph G, a colouring $\mathcal{C} = \{C_1, \ldots, C_k\}$, and a pair of vertices $u, v \in V$ that satisfy Theorem A.1 but not Theorem A.2, that is, finding $\mathcal{C}$ such that,

$$\begin{cases} \forall v \in V : |A_{C_{\varphi(v)}}(v)| - |A_{C_{\varphi(}}{}_i(v)| \leq 0 \\ \exists u, v \in V : |A_{C_{\varphi(v)}}(v)| - |A_{C_{\varphi(u)}}(v)| + |A_{C_{\varphi(u)}}(u)| - |A_{C_{\varphi(v)}}(u)| + 2 \cdot |A_{\{u\}}(v)| > 0 \end{cases} \tag{A.1}$$

If we assume, without loss of generality, that $|A_{C_{\varphi(v)}}(v)| = 1$, *i.e.*, v is involved in a conflict, and $f(\mathcal{C}) = 1$, then there may exist a graph G and a k-colouring of G such that $|A_{C_i}(v)| = 1 \ \forall i \in \Gamma$ and $\exists u : u \in A_V(v), \varphi(u) \neq \varphi(v)$ with $|A_{C_{\varphi(v)} \setminus \{v\}}(u)| = 0$. A graph G and a colouring of G with such properties represent a situation of local optimum for $\mathcal{N}_1$ but not for neighbourhood $\mathcal{N}_2$. Indeed, the Equation A.1 is satisfied by vertices u and v. In Figure A.1 we show that such a situation is possible by constructing a graph with 6 vertices and $k = 4$. There is no possible way to repair the conflict between v and x_1 in $\mathcal{N}_1$ but it is possible in $\mathcal{N}_2$ by changing simultaneously the colour of v and u.

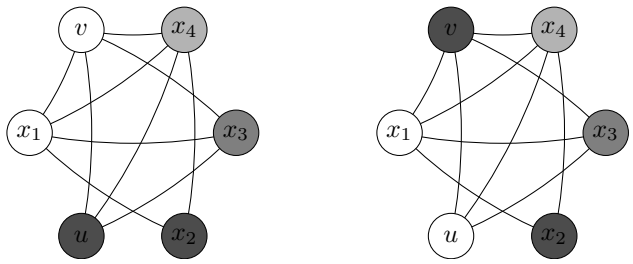


Figure A.1.: The graph on the left has the following infeasible colouring: $\mathcal{C}_1 = \{\{v, x_1\}, \{u, x_2\}, \{x_3\}, \{x_4\}\}$. In the swap neighbourhood it is possible to reach the feasible colouring $\mathcal{C}_2 = \{\{x_1, u\}, \{v, x_3, u\}, \{x_3\}, \{x_4\}\}$.

□

Theorem A.7 Iterative_Improvement$_{\mathcal{N}_1 \cup \mathcal{N}_2 \cup \mathcal{N}_3} \succ$ Iterative_Improvement$_{\mathcal{N}_1 \cup \mathcal{N}_2}$.

Proof. As for the previous theorem, the first part of the statement is verified because all $\mathcal{C}$ that are local optima for the neighbourhood $\mathcal{N}_1(\mathcal{C}) \cup \mathcal{N}_2(\mathcal{C}) \cup \mathcal{N}_3(\mathcal{C})$ are local optima also for the the neighbourhood $\mathcal{N}_1(\mathcal{C}) \cup \mathcal{N}_2(\mathcal{C})$. To show the second part it suffices to find a graph and infeasible colouring $\mathcal{C} = \{C_1, \ldots, C_k\}$ which is local optimum in the neighbourhood structure $\mathcal{N}_1 \cup \mathcal{N}_2$ but not in the neighbourhood structure $\mathcal{N}_3$. From Theorem A.3 this corresponds to finding a graph G and a colouring of it $\mathcal{C}$ such that, according to Theorems A.1, A.2, and A.3 there exists a set of vertices $u, v, w, x \in V$ which satisfies the following conditions:

$$\begin{cases} \begin{array}{rcl} |A_{C_{\varphi(u)}}(u)| - |A_{C_{\varphi(w)}}(u)| + |A_{C_{\varphi(w)}}(w)| - |A_{C_{\varphi(u)}}(w)| + 2 \cdot |A_{\{w\}}(u)| & \leq & 0 \\ |A_{C_{\varphi(v)}}(v)| - |A_{C_{\varphi(x)}}(v)| + |A_{C_{\varphi(x)}}(x)| - |A_{C_{\varphi(v)}}(x)| + 2 \cdot |A_{\{x\}}(v)| & \leq & 0 \\ |A_{C_{\varphi(v)}}(v)| - |A_{C_{\varphi(w)}}(v)| + |A_{C_{\varphi(w)}}(w)| - |A_{C_{\varphi(v)}}(w)| + 2 \cdot |A_{\{w\}}(v)| & \leq & 0 \\ |A_{C_{\varphi(u)}}(u)| - |A_{C_{\varphi(x)}}(u)| + |A_{C_{\varphi(x)}}(x)| - |A_{C_{\varphi(u)}}(x)| + 2 \cdot |A_{\{x\}}(u)| & \leq & 0 \\ |A_{C_{\varphi(x)}}(x)| - |A_{C_{\varphi(w)}}(x)| + |A_{C_{\varphi(w)}}(w)| - |A_{C_{\varphi(x)}}(w)| + 2 \cdot |A_{\{w\}}(x)| & \leq & 0 \\ |A_{C_{\varphi(u)}}(u)| - |A_{C_{\varphi(v)}}(u)| + |A_{C_{\varphi(v)}}(v)| - |A_{C_{\varphi(u)}}(v)| + 2 \cdot |A_{\{v\}}(u)| & \leq & 0 \end{array} \\ \underbrace{|A_{C_{\varphi(u)}}(u)| - |A_{C_{\varphi(w)}}(u)|}_{\leq 0} + \underbrace{|A_{C_{\varphi(v)}}(v)| - |A_{C_{\varphi(x)}}(v)|}_{\leq 0} + \\ \underbrace{|A_{C_{\varphi(w)}}(w)| - |A_{C_{\varphi(u)}}(w)|}_{\leq 0} + \underbrace{|A_{C_{\varphi(x)}}(x)| - |A_{C_{\varphi(v)}}(x)|}_{\leq 0} + \\ + 3 + |A_{\{v\}}(w)| + |A_{\{u\}}(x)| > 0 \end{cases} \quad \begin{array}{l} \text{(A.2a)} \\ \\ \\ \\ \\ \\ \text{(A.2b)} \end{array}$$

We can construct a graph and a k-colouring of it $\mathcal{C}$ that admit a solution to the system A.2 in the following way. We assume $f(\mathcal{C}) = 1$ and vertices $u, v \in V^c$. Furthermore we call $w, x \in V$ two vertices adjacent respectively of u and v (hence $|A_{\{u\}}(w)| = |A_{\{v\}}(x)| = 1$ and also $|A_{C_{\varphi(w)}}(w)| = |A_{C_{\varphi(x)}}(x)| = 0$). With these assumptions the first two inequalities of A.2a yield the conditions

$$-|A_{C_{\varphi(x)}}(v)| - |A_{C_{\varphi(v)}}(x)| \leq -2$$
$$-|A_{C_{\varphi(w)}}(v)| - |A_{C_{\varphi(v)}}(w)| \leq -2$$

which can be used in A.2b. The setting of values: $|A_{C_{\varphi(x)}}(v)| = 1$, $|A_{C_{\varphi(v)}}(x)| = 2$, $|A_{C_{\varphi(w)}}(v)| = 1$, $|A_{C_{\varphi(v)}}(w)| = 2$, and $|A_{\{v\}}(w)| = 1$, $|A_{\{u\}}(x)| = 1$ satisfies the inequality and it is not in contrast with any of the constraints A.2c-f. Such a situation is possible
In Figure A.2 we represent the graph and the colouring which results from the setting above with a size of the graph equal to 4. The reader can verify that there exists no improving neighbour in $\mathcal{N}_1$ and $\mathcal{N}_2$ but that the simultaneous swap of colours between respectively vertices v_1 and v_4 and v_2 and v_3 repairs the conflict present in the graph.

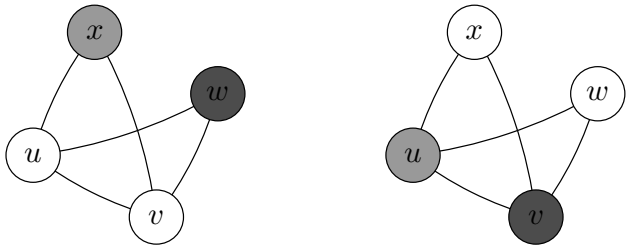


Figure A.2.: The graph on the left has the following infeasible colouring: $\mathcal{C}_1 = \{\{u, v\}, \{w\}, \{x\}\}$. In a pair swap exchange neighbourhood it is possible to reach the feasible colouring $\mathcal{C}_2 = \{\{u\}, \{v\}, \{w, x\}\}$.

□

Theorem A.8 $\mathsf{Iterative\_Improvement}_{\mathcal{N}_1 \cup \mathcal{N}_4} \succ \mathsf{Iterative\_Improvement}_{\mathcal{N}_1 \cup \mathcal{N}_2}$

Proof. We recall that, given a graph G, for each k-colouring $\mathcal{C} \in \mathcal{S}(G), k \geq \chi(G)$ it is $\mathcal{N}_2(\mathcal{C}) \subseteq N_4(\mathcal{C})$, as a swap can be represented as a degenerate cyclic exchange. Therefore

$\mathcal{N}_1(\mathcal{C}) \cup \mathcal{N}_2(\mathcal{C}) \subseteq \mathcal{N}_1(\mathcal{C}) \cup N_4(\mathcal{C})$ and every local optimum of the former is local optimum also of latter.
To prove the part (ii) of the Theorem (see Theorem A.6) it suffices to show that there exists a graph and a colouring $\mathcal{C}$ which is a local optimum for $\mathcal{N}_1(\mathcal{C}) \cup \mathcal{N}_2(\mathcal{C})$ but not for $\mathcal{N}_4(\mathcal{C})$. We restrict ourselves to consider a case with $m = 3$. We want to show that there may exist a set of vertices $v_1, v_2, v_3 \in V$ each belonging to a different colour class that satisfy the following conditions A.3a-A.3c due to Theorem A.4:

$$\begin{cases} \underbrace{|A_{C_{\varphi(v_1)}}(v_1)| - |A_{C_{\varphi(v_2)}}(v_1)|}_{\leq 0} + \underbrace{|A_{C_{\varphi(v_2)}}(v_2)| - |A_{C_{\varphi(v_1)}}(v_2)|}_{\leq 0} + 2 \cdot |A_{\{v_2\}}(v_1)| \leq 0 & \text{(A.3a)} \\ \underbrace{|A_{C_{\varphi(v_3)}}(v_3)| - |A_{C_{\varphi(v_1)}}(v_3)|}_{\leq 0} + \underbrace{|A_{C_{\varphi(v_1)}}(v_1)| - |A_{C_{\varphi(v_3)}}(v_1)|}_{\leq 0} + 2 \cdot |A_{\{v_3\}}(v_1)| \leq 0 & \text{(A.3b)} \\ \underbrace{|A_{C_{\varphi(v_2)}}(v_2)| - |A_{C_{\varphi(v_3)}}(v_2)|}_{\leq 0} + \underbrace{|A_{C_{\varphi(v_3)}}(v_3)| - |A_{C_{\varphi(v_2)}}(v_3)|}_{\leq 0} + 2 \cdot |A_{\{v_2\}}(v_3)| \leq 0 & \text{(A.3c)} \\ \underbrace{|A_{C_{\varphi(v_1)}}(v_1)| - |A_{C_{\varphi(v_2)}}(v_1)|}_{\leq 0} + |A_{\{v_1\}}(v_2)| + \underbrace{|A_{C_{\varphi(v_2)}}(v_2)| - |A_{C_{\varphi(v_3)}}(v_2)|}_{\leq 0} + |A_{\{v_3\}}(v_2)| + \underbrace{|A_{C_{\varphi(v_3)}}(v_3)| - |A_{C_{\varphi(v_1)}}(v_3)|}_{\leq 0} + |A_{\{v_1\}}(v_3)| > 0 & \text{(A.3d)} \end{cases}$$

If we assume that $f(\mathcal{C}) = 1$ and $v_1 \in V^c$, then it follows that $|A_{C_{\varphi(v_1)}}(v_1)| = 1$, $|A_{C_{\varphi(v_2)}}(v_2)| = 0$, and $|A_{C_{\varphi(v_3)}}(v_3)| = 0$. A possible way to satisfy A.3d is giving the following values to its components: $|A_{\{v_1\}}(v_2)| = |A_{\{v_1\}}(v_3)| = |A_{C_{\varphi(v_2)}}(v_1)| = |A_{C_{\varphi(v_1)}}(v_3)| = 1$. Then setting $|A_{C_{\varphi(\{v_3\})}}(v_2)| = |A_{\{v_3\}}(v_2)|$ or $|A_{C_{\varphi(\{v_3\})}}(v_2)| = |A_{\{v_3\}}(v_2)| + 1$ the condition is satisfied.
This configuration does not contrast with the other conditions. Indeed we note that there is always the fourth term in each equation A.3 which can be properly set to satisfy the equation. For example, a possible solution is given by setting $|A_{C_{\varphi(\{v_1\})}}(v_2)| = |A_{C_{\varphi(\{v_3\})}}(v_1)| = 2$ and all free terms in condition A.3c equal to zero. Such a situation is possible and it is represented in Figure A.3. The graph and the proposed k-colouring respect all conditions of local optimality in $\mathcal{N}_1$ and $\mathcal{N}_2$ while a cyclic exchange of the vertices v_1, v_2, v_3 repairs the conflict present in the colouring.

□

Theorem A.9 Iterative_Improvement$_{\mathcal{N}_2 \cup \mathcal{N}_5} \succ$ Iterative_Improvement$_{\mathcal{N}_1 \cup \mathcal{N}_2}$.

Proof. We recall that, given a graph G, for each k-colouring $\mathcal{C} \in \mathcal{S}(G), k \geq \chi(G)$ it is $\mathcal{N}_1(\mathcal{C}) \subseteq N_5(\mathcal{C})$, as a one-exchange can be represented as a degenerate case of path exchange. Therefore $\mathcal{N}_1(\mathcal{C}) \cup \mathcal{N}_2(\mathcal{C}) \subseteq \mathcal{N}_2(\mathcal{C}) \cup N_5(\mathcal{C})$ and every local optimum of the former is local optimum also of latter.
To prove the part (ii) of the Theorem it suffices to find a colouring $\mathcal{C}$ which is a local optimum for $\mathcal{N}_1(\mathcal{C})$ and $\mathcal{N}_2(\mathcal{C})$ but not for $\mathcal{N}_5(\mathcal{C})$. For $m = 3$ and according to Theorem

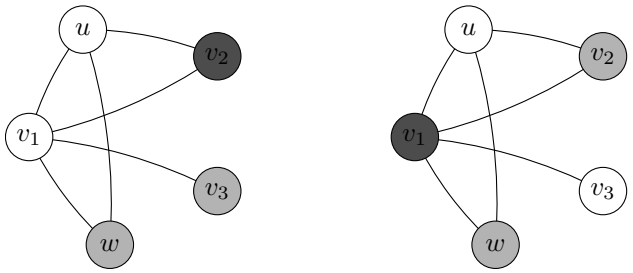


Figure A.3.: The graph on the left has the following infeasible colouring: $\mathcal{C}_1 = \{\{v_1, u\}, \{v_2\}, \{v_3, w\}\}$. In a cyclic exchange of $V' = \{v_1, v_2, v_3\}$ it is possible to reach the feasible colouring $\mathcal{C}_2 = \{\{v_3\}, \{v_1, u\}, \{w, v_2\}$.

A.2 and Theorem A.5, $\mathcal{C}$ is not a local optimum for $\mathcal{N}_5$ if there exists at least a set of vertices $v_1, v_2, v_3 \in V, \varphi(v_1) \neq \varphi(v_2) \neq \varphi(v_3)$ such that:

$$\begin{cases} \underbrace{|A_{C_{\varphi(v_1)}}(v_1)| - |A_{C_{\varphi(v_2)}}(v_1)|}_{\leq 0} + \underbrace{|A_{C_{\varphi(v_2)}}(v_2)| - |A_{C_{\varphi(v_1)}}(v_2)|}_{\leq 0} + 2 \cdot |A_{\{v_2\}}(v_1)| \leq 0 & \text{(A.4a)} \\ \underbrace{|A_{C_{\varphi(v_3)}}(v_3)| - |A_{C_{\varphi(v_1)}}(v_3)|}_{\leq 0} + \underbrace{|A_{C_{\varphi(v_1)}}(v_1)| - |A_{C_{\varphi(v_3)}}(v_1)|}_{\leq 0} + 2 \cdot |A_{\{v_3\}}(v_1)| \leq 0 & \text{(A.4b)} \\ \underbrace{|A_{C_{\varphi(v_2)}}(v_2)| - |A_{C_{\varphi(v_3)}}(v_2)|}_{\leq 0} + \underbrace{|A_{C_{\varphi(v_3)}}(v_3)| - |A_{C_{\varphi(v_2)}}(v_3)|}_{\leq 0} + 2 \cdot |A_{\{v_2\}}(v_3)| \leq 0 & \text{(A.4c)} \\ \underbrace{|A_{C_{\varphi(v_1)}}(v_1)| - |A_{C_{\varphi(v_2)}}(v_1)|}_{\leq 0} + \underbrace{|A_{C_{\varphi(v_2)}}(v_2)| - |A_{C_{\varphi(v_3)}}(v_2)|}_{\leq 0} + |A_{\{v_1\}}(v_2)| > 0 & \text{(A.4d)} \end{cases}$$

If we assume, without loss of generality, that $f(\mathcal{C}) = 1$ and $v_1 \in V^c$, then $|A_{C_{\varphi(v_1)}}(v_1)| = 1$ and $|A_{C_{\varphi(v)}}(v)| = 0$ for any other vertex in the graph, except another vertex $u \in V$. Using the result of Theorem A.1 in Equation A.4d, as indicated by the brackets, it follows that the condition A.4d is satisfied if, for example we set $|A_{C_{\varphi(v_3)}}(v_2)| = 0$ and $|A_{\{v_1\}}(v_2)| + 1 > |A_{C_{\varphi(v_2)}}(v_1)|$. Note that if $|A_{C_{\varphi(v_3)}}(v_2)| = 0$ then $|A_{v_2}(v_3)| = 0$ and $|A_{C_{\varphi(v_2)}}(v_3)| = 0$, hence A.4c is satisfied. Moreover, by setting $|A_{C_{\varphi(v_2)}}(v_1)| = 1$ the inequality A.4a and A.4d are also both satisfied. Finally the condition A.4a is satisfied by setting $|A_{v_3}(v_1)| = 1$ and consequently also $|A_{C_{\varphi(v_1)}}(v_3)| = |A_{C_{\varphi(v_3)}}(v_1)| = 1$. Therefore a solution for the system A.4 exists. In Figure A.4 we show that such solution correspond to a possible situation. We observe that no one-exchange or swap exchange involving any of the vertices in the graph leads to an improvement in the colouring while a path exchange involving vertices v_1, v_2, and v_3 solves the only conflict present.

□

Theorem A.10 $\mathsf{Iterative\_Improvement}_{\mathcal{N}_4 \cup \mathcal{N}_5} \succ \mathsf{Iterative\_Improvement}_{\mathcal{N}_2 \cup \mathcal{N}_5}$.

Proof. Since for all colourings $\mathcal{C}$ it is $\mathcal{N}_2(\mathcal{C}) \cup \mathcal{N}_5(\mathcal{C}) \subseteq \mathcal{N}_4(\mathcal{C}) \cup \mathcal{N}_5(\mathcal{C})$ the first part of the theorem is trivially true. We show now that it is possible to construct a graph and a

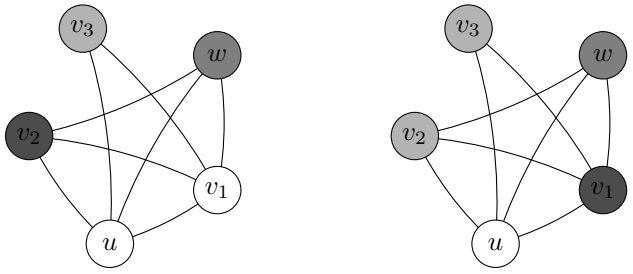


Figure A.4.: The graph on the left has the following infeasible colouring: $\mathcal{C}_1 = \{\{v_2\}, \{v_1, u\}, \{w\}, \{v_3\}\}$. In a path exchange neighbourhood it is possible to reach the feasible colouring $\mathcal{C}_2 = \{\{v_2, v_3\}, \{w\}, \{v_1\}, \{u\}\}$ by applying a path exchange of the vertices $V' = \{v_1, v_2, v_3\}$.

k-colouring $\mathcal{C}$ such that the configuration is a local optimum for $\mathcal{N}_2 \cup \mathcal{N}_5$ but not for $\mathcal{N}_4$, that is, there exists no swap exchange and path exchange that yield a better $\mathcal{C}$ but there exists a cyclic exchange of a subset of vertices $V' \subseteq V$, each vertex belonging to a different colour class, which does.
For showing this we have to show that the following system, obtained by the application of the results of Theorems A.2, A.5, and A.4, admits a set of vertices $V' = \{v_1, \ldots, v_m\}$ as solution:

$$\begin{cases} \forall v_i, v_j \in V' : \\ |A_{C_{\varphi(v_i)}}(v_i)| + |A_{C_{\varphi(v_j)}}(v_j)| - |A_{C_{\varphi(v_j)}}(v_i)| - |A_{C_{\varphi(v_i)}}(v_j)| + 2 \cdot |A_{\{v_j\}}(v_i)| \leq 0 & \text{(A.5a)} \\ \sum_{i=1}^{m-1} |A_{C_{\varphi(v_i)}}(v_i)| - \sum_{i=1}^{m-1} |A_{C_{\varphi(v_{i+1})}}(v_i)| + \sum_{i=1}^{m-2} |A_{\{v_i\}}(v_{i+1})| \leq 0 & \text{(A.5b)} \\ \sum_{i=1}^{m} |A_{C_{\varphi(v_i)}}(v_i)| - \sum_{i=2}^{m} |A_{C_{\varphi(v_i)}}(v_{i-1})| + \sum_{i=2}^{m} |A_{\{v_{i-1}\}}(v_i)| + \\ \qquad - |A_{C_{\varphi(v_1)}}(v_m)| + |A_{\{v_m\}}(v_1)| > 0 & \text{(A.5c)} \end{cases}$$

The inequality A.5c can be rewritten and reordered as

$$\overbrace{\sum_{i=1}^{m-1} |A_{C_{\varphi(v_i)}}(v_i)| - \sum_{i=1}^{m-1} |A_{C_{\varphi(v_{i+1})}}(v_i)| + \sum_{i=1}^{m-2} |A_{\{v_i\}}(v_{i+1})|}^{\leq 0 \text{ (A.5b)}} + \underbrace{|A_{C_{\varphi(v_m)}}(v_m)| - |A_{C_{\varphi(v_1)}}(v_m)|}_{\leq 0} + |A_{\{v_{m-2}\}}(v_{m-1})| + |A_{\{v_m\}}(v_1)| > 0$$

Hence at least one of the two terms $|A_{v_{m-2}}(v_{m-1})|$ and $|A_{v_m}(v_1)|$ must be equal to one. We assume without loss of generality that $f(\mathcal{C}) = 1$ and $|A_{C_{\varphi(v_m)}}(v_m)| = 1$. The term $|A_{v_m}(v_1)|$ does not appear in condition A.5b. If we set it equal to 1 then $|A_{C_{\varphi(v_1)}}(v_m)| \geq 1$ and $|A_{C_{\varphi(v_m)}}(v_1)| \geq 1$. For condition A.5a it must then be

$$|A_{C_{\varphi(v_m)}}(v_m)| - |A_{C_{\varphi(v_1)}}(v_m)| + |A_{C_{\varphi(v_1)}}(v_1)| - |A_{C_{\varphi(v_m)}}(v_1)| + 2 \cdot |A_{\{v_1\}}(v_m)| \leq 0$$

and hence $|A_{C_{\varphi(v_m)}}(v_1)| \geq 2$. Apparently no contrast emerges among the conditions that is easy to detect. We also verify that a solution to the system exists which is not degenerate (hence: $m \neq 2$ and the graph is not disconnected, *i.e.*, the condition A.5b is not satisfied just by setting all terms equal to zero). In Figure A.5 we show a possible configuration in which the set $\{v_1, v_2, v_3\}$ is a solution for the system A.5. It is $|A_{v_3}(v_3)| = 1$ and $|A_{C_{\varphi(v_3)}}(v_1)| = 2$, which we already showed is a feasible configuration for the system and satisfies the condition A.5c. Moreover, the following conditions, derived from conditions A.5a and A.5b, are also satisfied. It is finally easy to see that, for the specific configuration of Figure A.5, the same property holds for all possible combinations of 3 vertices belonging to different colour classes.

$$\begin{cases} A_{C_{\varphi(v_1)}}(v_1) - A_{v_2}(v_1) + A_{C_{\varphi(v_2)}}(v_2) - A_{v_1}(v_2) + 2A_{v_1}(v_2) & \leq & 0 \\ A_{C_{\varphi(v_1)}}(v_1) - A_{C_{\varphi(v_2)}}(v_1) + A_{C_{\varphi(v_2)}}(v_2) - A_{C_{\varphi(v_3)}}(v_2) + A_{v_1}(v_2) & \leq & 0 \end{cases}$$

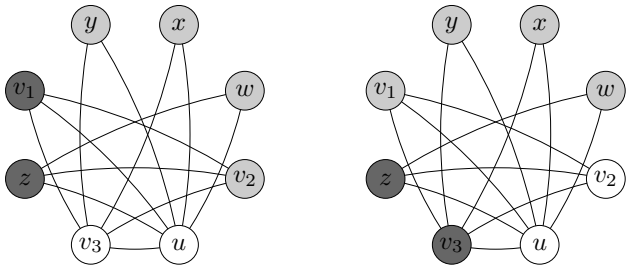


Figure A.5.: The graph on the left has the following infeasible colouring: $\mathcal{C}_1 = \{\{v_8, v_1\}, \{v_2, v_3\}, \{v_4, v_5, v_6, v_7\}\}$. In a cyclic exchange neighbourhood a set $V' = \{v_8, v_4, v_2\}$ can be found that makes the colouring feasible: $\mathcal{C}_2 = \{\{v_1, v_2\}, \{v_3, v_4\}, \{v_5, v_6, v_7, v_8\}\}$.

□

Theorem A.11 Iterative_Improvement$_{\mathcal{N}_4 \cup \mathcal{N}_5} \succ$ Iterative_Improvement$_{\mathcal{N}_1 \cup \mathcal{N}_4}$.

Proof. For all $\mathcal{C}$ it is $\mathcal{N}_1(\mathcal{C}) \subseteq \mathcal{N}_5(\mathcal{C})$. Hence, as for the previous theorems to show that the statement is correct it suffices to show that there exists a graph and a colouring k such that no one-exchange or cyclic exchange that yield a better $\mathcal{C}$ but there exists a path exchange of a subset of vertices $V' \subseteq V$, each vertex belonging to a different colour class, which does.

$$\begin{cases} \forall j \in \Gamma, v \in V : \; |A_{C_{\varphi(v)}}(v)| - |A_j(v)| \leq 0 & \text{(A.6a)} \\ \sum_{i=1}^{m} |A_{C_{\varphi(v_i)}}(v_i)| - \sum_{i=1}^{m-1} |A_{C_{\varphi(v_{i+1})}}(v_i)| + \sum_{i=1}^{m-1} |A_{\{v_i\}}(v_{i+1})| + \\ \qquad - |A_{C_{\varphi(v_1)}}(v_m)| + |A_{\{v_m\}}(v_1)| \leq 0 & \text{(A.6b)} \\ \sum_{i=1}^{m-1} |A_{C_{\varphi(v_i)}}(v_i)| - \sum_{i=1}^{m-1} |A_{C_{\varphi(v_{i+1})}}(v_i)| + \sum_{i=1}^{m-2} |A_{\{v_i\}}(v_{i+1})| > 0 & \text{(A.6c)} \end{cases}$$

The condition A.6b is obtained by Theorem A.4 simply by opportunely changing the indexes.

We show that the set of vertices $V' = \{v_1, v_2, v_3\}$ and the configuration from Figure A.6 may be a feasible solution to the system A.6.

The case is $m = 3$, therefore, we can rewrite the system A.6 as

$$\begin{cases} \overbrace{|A_{C_{\varphi(v_1)}}(v_1)| - |A_{C_{\varphi(v_2)}}(v_1)|}^{\leq 0} + \overbrace{|A_{C_{\varphi(v_2)}}(v_2)| - |A_{C_{\varphi(v_3)}}(v_2)|}^{\leq 0} + \\ |A_{C_{\varphi(v_3)}}(v_3)| + |A_{\{v_1\}}(v_2)| + |A_{\{v_2\}}(v_3)| - |A_{C_{\varphi(v_1)}}(v_3)| + |A_{\{v_1\}}(v_3)| \leq 0 \\ \overbrace{|A_{C_{\varphi(v_1)}}(v_1)| - |A_{C_{\varphi(v_2)}}(v_1)|}^{\leq 0} + \overbrace{|A_{C_{\varphi(v_2)}}(v_2)| - |A_{C_{\varphi(v_3)}}(v_2)|}^{\leq 0} + |A_{\{v_1\}}(v_2)| > 0 \end{cases}$$

It is easy to verify that the set $\{v_1, v_2, v_3\}$ satisfies the system. Given the symmetry of the graph and its colouring, the set $\{u, v_3, x\}$ is also a solution. It remains to verify that no one-exchange and cyclic exchange can repair the only conflict present in the configuration (u, v_1). In order to do this we first note that either vertex u or vertex v_3 must be involved in the exchange. No improving one-exchange is possible for these two vertices, as the condition A.6a holds for both of them (they are indeed connected to vertices with all three usable colours). Any cyclic exchange, by definition, should involve the three colour classes, therefore the only candidate to check is $V' = \{v_1, v_2, v_3\}$, as the other three possibilities $\{v_1, x, v_3\}$, $\{u, v_2, v_3\}$, and $\{u, x, v_3\}$ are just symmetric configurations. Being V' solution of the system we already checked that this set verifies the condition A.6b and hence cannot be an improving cyclic exchange.

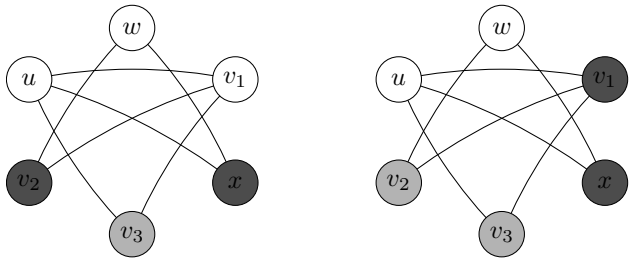


Figure A.6.: The graph on the left has the following infeasible colouring: $\mathcal{C}_1 = \{\{u, v_1, w\}, \{v_3\}, \{v_2, x\}\}$. The feasible colouring $\mathcal{C}_2 = \{\{u, w\}, \{v_1\}, \{v_2, v_3\}\}$ can be reached by a path exchange of the vertices $V' = \{v_1, v_2, v_3\}$.

□

A.3. Summary of results and discussion

We can summarise the results as follows:

$$\mathsf{Iterative\_Improvement}_{\mathcal{N}_1 \cup \mathcal{N}_2} \succ \mathsf{Iterative\_Improvement}_{\mathcal{N}_1}$$

$$\mathsf{Iterative\_Improvement}_{\mathcal{N}_1 \cup \mathcal{N}_2 \cup \mathcal{N}_3} \succ \mathsf{Iterative\_Improvement}_{\mathcal{N}_1 \cup \mathcal{N}_2}$$

$$\mathsf{Iterative\_Improvement}_{\mathcal{N}_4 \cup \mathcal{N}_5} \succ \mathsf{Iterative\_Improvement}_{\mathcal{N}_1 \cup \mathcal{N}_4} \succ \mathsf{Iterative\_Improvement}_{\mathcal{N}_1 \cup \mathcal{N}_2}$$

$$\mathsf{Iterative\_Improvement}_{\mathcal{N}_4 \cup \mathcal{N}_5} \succ \mathsf{Iterative\_Improvement}_{\mathcal{N}_2 \cup \mathcal{N}_5} \succ \mathsf{Iterative\_Improvement}_{\mathcal{N}_1 \cup \mathcal{N}_2}$$

We presented a combinatorial dominance analysis which clearly indicates that the extension of the canonical neighbourhood for local search by the union of cyclic and path exchanges allows to attain better quality solutions. Other similar works on dominance analysis for heuristics are known in the literature on other problems (Gutin et al., 2003, 2002; Skiena and Berend, 2004). In particular, contrary to the result found on the TSP, exponentially searchable neighbourhoods dominate polynomially searchable ones. We conclude that the combination of cyclic and path exchanges is the most appealing combination of neighbourhoods among those studied. Different to the work of (Gutin et al., 2002) we restricted ourselves, however, to a relative rather than quantitative analysis between the algorithms proposed. Even with a very large scale neighbourhood there exist very simple cases, as those of Figure A.7, that are infeasible colourings and local optima in all neighbourhoods combinations discussed.

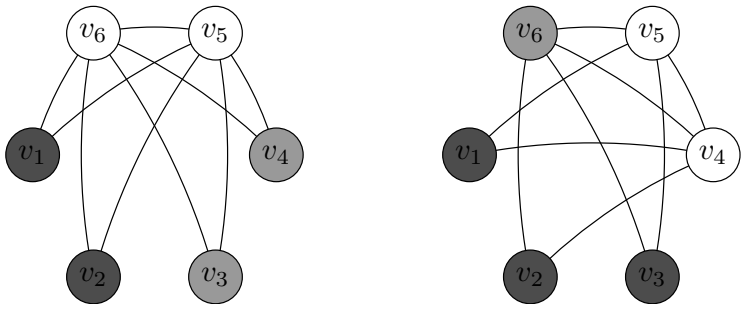


Figure A.7.: Two situations in which no improving neighbouring colouring exists in any of the neighbourhood structures introduced in the paper. The colouring on the left is $\mathcal{C}_1 = \{\{v_1, v_2\}, \{v_3, v_4\}, \{v_5, v_6\}\}$ while the one on the right is $\mathcal{C}_2 = \{\{v_1, v_2, v_3\}, \{v_4, v_5\}, \{v_6\}\}$.

Appendix B.

On the Behaviour of the Statistical Tests

In this appendix, we provide details and comparisons of the statistical tests defined in Chapter 3 and used in the analysis of experiments presented throughout this thesis. We focus on the scenario *"several runs on various instances"*, which corresponds to a two way factorial layout, and we consider tests based on the parametric ANOVA, on Friedman's ranks, and on permutations of data (all these tests are described in Section 3.6.6). Although ANOVA and Friedman's tests are well known, an assessment of the permutation tests used in this thesis is needed because (i) it is opportune to validate our implementation of these tests; (ii) the USP method for factorial layouts with few replicates has never been tested before; (iii) there are different opinions concerning the appropriate permutation methods that may be used for particular tests when the design includes more than one factor (ter Braak, 1992, suggested permutation of residuals, while Gonzalez and Manly, 1998, suggested unrestricted, *i.e.*, not synchronised, permutations of raw data), hence, we need to corroborate our choice; iv) we are not aware of any comparison with parametric tests on distributions of data similar to those hypothesised for stochastic optimisers. We intend to assess the procedures of permutation tests for simultaneous confidence intervals in the all-pairwise comparisons that have been introduced in this thesis. Simulation studies for the assessment and comparison of permutation tests with other tests are widely used in Applied Statistics. Similar examples to the one described in this Appendix are reported by Anderson and ter Braak (2003) for multi-factorial analysis of variance and by Finos et al. (2003) for multiple comparisons test procedures. In their study, Anderson and ter Braak (2003) argue that exchanging observations is appealing due to the intuitive wisdom underlying this principle, but that a procedure that is likely of attaining better results is exchanging error residuals. Nevertheless, they do not consider synchronised permutations.

In our analysis, we will distinguish the two procedures for generating synchronised permutations: *constrained synchronised permutations* (CSP) and *unconstrained synchronised permutations* (USP), which were mentioned in Section 3.6.6 on page 60. We first give further details about their implementation that were not given in the text. We then present the results of the simulation studies for comparing the type I error rate and the statistical power on global hypothesis testing for a two way factorial layout. Finally, we turn to all-pairwise comparisons and present the results of simulation studies on the same two way layout.

B.1. Implementation details

Constrained Synchronised Permutation. CSP procedures allow to state easily the number of possible permutations. In a $h \times k$ scenario with r replicates, the number of permutations that yield a different value of the test statistics is $\binom{2r}{r}/2$, the reason for

the division by 2 being that the test statistic of Equation 3.16, on page 60, is symmetric. Hence, the minimum α-value attainable from an experiment for a two-sided tests corresponds to $[\binom{2r}{r}/2]^{-1}$: for $r = 3$, the minimum α value is 0.1, which is likely to be too big. Therefore, experiments using CSP should collect at least 4 replicates that allow for a minimum α value of about 0.03. When CSP are not sufficient the choice should be for USP that enlarge the number of possible configurations such that the exact approach becomes infeasible but the minimum α value decreases.

Unconstrained Synchronised Permutations. In Algorithm 3.5, page 62, we left open the issue of how to implement a Conditional Monte Carlo (CMC) sampling procedure without-replacement in which all different permutations are equally likely to appear.[1] The crucial point is the determination of the the number of exchanges ν^*, $\nu^* \in \{0, \ldots, r\}$, to perform at each sample. Assigning to each value ν^* the same probability to appear is not correct rather the probability must be determined by the number of permutations that are possible for each specific value of ν^*. Hence, the probability distribution of ν^* is given by

$$P(\nu^* = i) = \Pi_i / \Pi \qquad \forall i \in \{0, \ldots, r\}$$

where Π is the total number of distinct permutations and Π_i is the number of distinct permutations with $\nu^* = i$. We have

$$\Pi_i = \binom{r}{i}^{2hc} \quad \forall i \in \{0, \ldots, r\} \qquad \Pi = \sum_{i=0}^{r} \Pi_i = \sum_{i=0}^{r} \binom{r}{i}^{2hc}$$

where $c = \binom{k}{2}$ is the number of different levels pairs in the tested factor.

The complete characterisation of the discrete probability distribution for ν^* can be avoided by noting that the test statistic and the binomial coefficient are symmetric. We can, therefore, restrict ourselves to consider half of the possible values for ν^* in the following way. We distinguish two cases:

Case r is odd. Then we have $P(\nu^* \leq \frac{r-1}{2}) = P(\nu^* \geq \frac{r+1}{2}) = 0.5$. Hence, we can restrict ourselves to consider $\nu^* \in \{0, \ldots, \frac{r-1}{2}\}$ and construct the corresponding probabilities from

$$\Pi_i = \binom{r}{i}^{2hc} \qquad \Pi = \sum_{i=0}^{\frac{r-1}{2}} \Pi_i = \sum_{i=0}^{\frac{r-1}{2}} \binom{r}{i}^{2bc}$$

Case r is even. Then we have $P(\nu^* = \frac{r}{2}) + 2P(\nu^* < \frac{r}{2}) = 1$. Hence, we can restrict ourselves to consider $\nu^* \in \{0, \ldots, \frac{r}{2}\}$ by noting that $P(\nu^* = \frac{r}{2}) = \frac{\Pi_{\frac{r}{2}}}{2\sum_{i=1}^{\frac{r}{2}-1} \Pi_i + \Pi_{\frac{r}{2}}}$ and constructing the probability distributions from

$$\Pi_i = \binom{r}{i}^{2hc} \quad \forall i \in \{0, \ldots, \frac{r}{2} - 1\} \qquad \Pi_{\frac{r}{2}} = \left[\binom{r}{r/2}^{2b} \Big/ 2 \right]^c$$

$$\Pi = \sum_{i=0}^{\frac{r}{2}} \Pi_i = \sum_{i=0}^{\frac{r}{2}-1} \binom{r}{i}^{2bc} + \left[\binom{r}{r/2}^{2b} \Big/ 2 \right]^c$$

[1] Our proposed solution is the outcome of a joint work with D. Basso, Department of Statistics, University of Padova.

Coding details. We coded the permutation tests in the statistical environment R with external calls to C code for the heaviest computations. The generation of different combinations of interest in the exact implementation of CSP is done in R by a procedure that generates them by minimal changes (this is possible using Gray Code, Press et al., 1992). This entails the further advantage that, by considering only the first half of $\binom{2r}{r}$ combinations, it is possible to characterise one half of the symmetric statistic distribution.

In USP, the computation of Π requires some attention, as it can easily cause numeric overflow in computer programs: indeed, already for a 3×5 design with $r = 5$ we have $\Pi \approx 4.1 \cdot 10^{14}$, which is lager than what can be represented by a 32-bits word since $2^{32} \approx 4.3 \cdot 10^{9}$. For the same design with 6 replicates we have $\Pi \approx 10^{39}$, which is larger than what can be represented by a 64-bits word $2^{6} \approx 5.7 \cdot 10^{19}$. In our implementation we computed the probability distribution of ν^* in R before passing it to the C code. However for designs with more than 3 replicates CSP should be enough and USP is not needed.

Attention should be given also to the choice of the sample size in CMC sampling. In particular, in all-pairwise comparisons, where the comparison-wise α-value could be adjusted, it must be checked that the new value remains larger than the minimal attainable α-value of the test. Sizes of 1000 and 2000 have been sufficient for the experiments of this thesis.

The pseudo-random generator adopted in our implementation is the one originally suggested by L'Ecuyer (1988) (and also reported in Press et al., 1992) which, besides being machine independent, has period larger than $2 \cdot 10^{18}$, thus providing us with enough guarantees that no bias is introduced due to repetition of values in the pseudo-random sequence.

B.2. A simulation study for general hypothesis testing

B.2.1. Comparison of Type I Error Rates

Statistical theory tells us that the type I error rates are close to α as long as model assumptions are nearly correct. By means of repeated simulation of data we estimate the error rates of four statistical tests: ANOVA, rank-based, and the two permutation procedures CSP and USP.

We generate data under the assumption that the underlying populations from which observations are drawn are all equal in mean, variance, and distributions, that is, the null hypothesis is true. We consider four different underlying distributions for the generated data giving rise to four simulation cases. The distributions we consider are the *normal*, the *Student's t*, the *exponential*, and the *Weibull* ones. The *normal* distributions serve us to validate the permutation test procedures when parametric tests are valid while the *Student's t* distributions exemplify the case of distribution of results with heavier tails. The *exponential* distribution and the *Weibull* distribution represent the case of two asymmetric distributions and may be likely the case with stochastic optimisers. In particular, some authors (McRoberts, 1971; Dannenbring, 1977; Golden and Alt, 1979; Smith and Sucur, 1996) noted that the theoretical distribution that may approximate the results of SLS algorithms is the Weibull distribution. This is because each run of a SLS algorithm may be seen as a sampling procedure through local optima, the final result begin the best among the visited local optima. These assumptions are at the basis of *extreme value theory* ac-

cording to which the extreme values of random samples are characterised by a Weibull distribution.

More in specific, each observation of the simulated data is represented by $X_{ij} = \mu + \tau_i + \theta_j + (\tau\theta)_{ij} + \epsilon_{ij}$. Under the null hypothesis all effects are set to zero and also the true mean $\mu = 0$. The remaining model is therefore $X_{ij} = \epsilon_{ij}$ where the error terms ϵ_{ij} are distributed as follows.

Normal distribution (Norm): experimental errors are normally distributed with zero mean and unit variance.

Exponential distribution (Exp): experimental errors are exponentially distributed with unit mean.

Student's *t* distribution (t3): experimental errors have Student's t distribution with 3 degrees of freedom and are divided by $\sqrt{3}$.

Weibull distribution (Weib): experimental errors have Weibull distribution with shape equal to 1.2 and scale equal to 10 and are divided by 8.

In Figure B.1, we give an example of the smoothed probability density from data drawn from the four theoretical distributions. In Figure B.2, we show the effect of the same data on the typical diagnostic plots to detect the normality of residuals. Note that the diagnostic plots of residuals reflect the distributions of the observations and remain not normal even if the number of replicates is very high, 100 in the case of the figure. The implication of the central limit theorem concerns only the estimation of parameters obtained by linear combination from the data and not the data itself.

For our simulation study we considered the scenario of a 3×5 design (*i.e.*, $k = 3$ and $h = 5$) with $r = \{3, 5, 10\}$ replicates. For each different r and each different underlying distribution of errors we simulated 1000 independent data sets and observed the rate of rejection of H_0. Results are reported in Table B.1. In the first column on the left the nominal α-values are reported. In the case of $r = 5$ these values are multiples of the minimum α value attainable for the CSP test. With only 3 replicates the minimum α value is still too high, therefore we do not report its results. With 5 replicates the CSP test is exact in the sense that all 126 possible permutations yielding a different value for the test statistic are generated. With 10 replicates a CMC sampling procedure is used with a sample size of 1000. For USP, instead, a CMC sampling procedure of the same size is always applied. The effects relative to the factor with three levels are indicated by τ.

Comments The first observation is that ANOVA exhibits a surprisingly good behaviour under all the four tested cases and even with few replicates. A similar consideration is valid also for the Friedman's test that seems to increase slightly his error rate only with the Student's *t* distribution. The robustness of these two test procedures with respect to the violation of the assumption of normality distribution of data is known (Montgomery, 2000). The comparison with the plots of Figure B.2 may be useful for further applications. Moreover, we recall that if the diagnostic plots do not resemble those of the figure, it may still be possible to reach them after some reasonable transformation of the data. Permutation tests exhibit a worse behaviour. In particular, they control less the error rate for the factor with 5 levels and for the interactions. USP maintain coherent his behaviour over the three different r values. Already with 5 replicates, however, CSP are enough as they behave not differently than with 5 replicates and actually are better than USP.

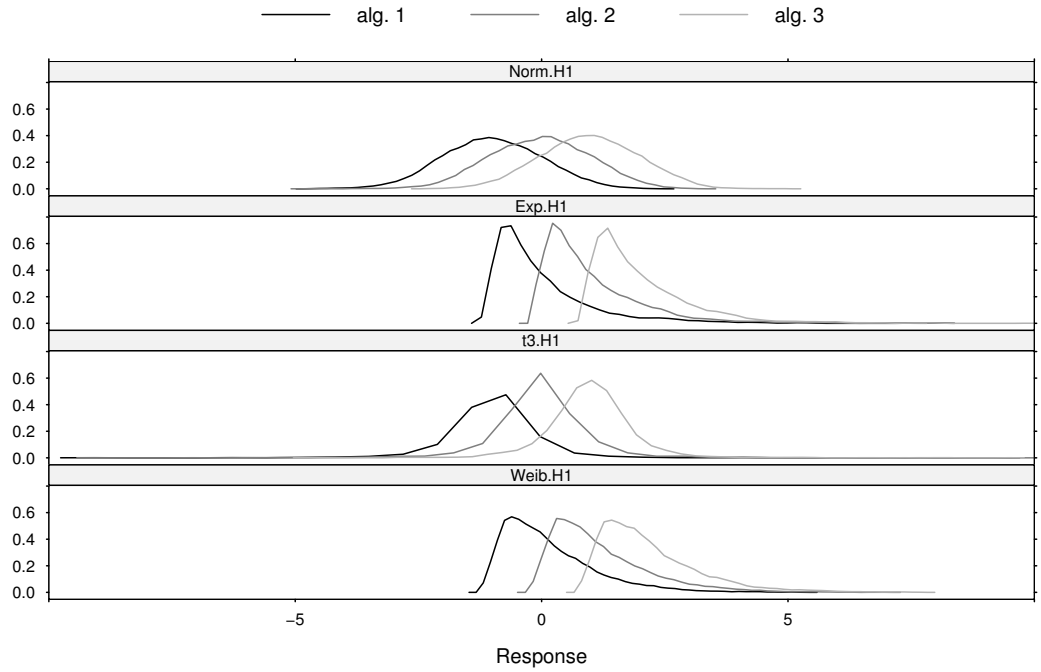


Figure B.1.: Smoothed distributions for data sampled from the four theoretical distributions normal, exponential, Student's t, and Weibull. A shift equal to 1 is assigned to the values of the distributions in order to make them distinguishable. The null hypothesis is, therefore, violated in the case depicted.

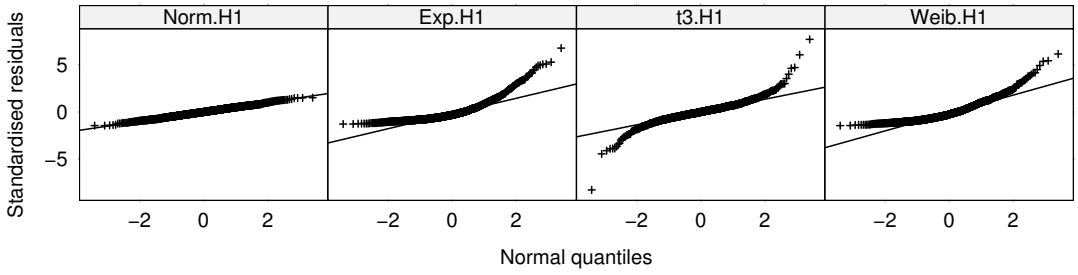


Figure B.2.: Diagnostic plots for checking the normality assumption of residual errors implicit in parametric ANOVA. The data are those of Figure B.1.

B.2.2. Comparison of powers

Among tests having equal type I error rates, the most powerful one is preferable. To assess the power of each test we generated data under the condition that the null hypothesis is violated. To this end we consider again the model $X_{ij} = \mu + \tau_i + \theta_j + (\tau\theta)_{ij} + \epsilon_{ij}$ and set $\mu = 0$. As praxis in statistical simulations we conform to the restriction for fixed effects which entails that the sum of the individual treatment effects across all levels of a fixed factor equals zero. Accordingly, the terms τ_i,θ_j,$(\tau\theta)_{ij}$ are obtained as indicated by the experimental matrix of Table B.2. The error terms ϵ_{ij} are then distributed according to

3 Replicates	Norm.H_0				Exp.H_0				t3.H_0				Weib.H_0		
	τ	θ	$(\tau\theta)$		τ	θ	$(\tau\theta)$		τ	θ	$(\tau\theta)$		τ	θ	$(\tau\theta)$
ANOVA															
0.01	0.008	0.006	0.005		0.009	0.011	0.021		0.003	0.009	0.007		0.012	0.015	0.012
0.05	0.053	0.052	0.037		0.059	0.048	0.064		0.040	0.041	0.047		0.055	0.056	0.056
0.1	0.109	0.117	0.093		0.103	0.097	0.099		0.083	0.075	0.089		0.112	0.116	0.110
Rank															
0.01	0.007	0.003	–		0.010	0.005	–		0.009	0.008	–		0.009	0.008	–
0.05	0.047	0.044	–		0.044	0.052	–		0.041	0.044	–		0.056	0.049	–
0.1	0.109	0.109	–		0.095	0.114	–		0.085	0.095	–		0.110	0.099	–
USP															
0.01	0.018	0.062	0.079		0.023	0.063	0.099		0.019	0.054	0.085		0.027	0.071	0.103
0.05	0.064	0.153	0.164		0.074	0.121	0.162		0.058	0.110	0.149		0.070	0.137	0.171
0.1	0.121	0.207	0.217		0.120	0.170	0.210		0.109	0.169	0.209		0.122	0.174	0.227
5 Replicates	Norm.H_0				Exp.H_0				t3.H_0				Weib.H_0		
	τ	θ	$(\tau\theta)$		τ	θ	$(\tau\theta)$		τ	θ	$(\tau\theta)$		τ	θ	$(\tau\theta)$
ANOVA															
0.016	0.012	0.016	0.016		0.014	0.012	0.010		0.012	0.009	0.010		0.007	0.011	0.014
0.055	0.046	0.051	0.061		0.046	0.050	0.041		0.050	0.036	0.038		0.056	0.049	0.052
0.11	0.103	0.108	0.124		0.105	0.107	0.101		0.093	0.089	0.086		0.108	0.100	0.104
Rank															
0.016	0.014	0.015	–		0.016	0.012	–		0.013	0.009	–		0.017	0.012	–
0.055	0.050	0.052	–		0.052	0.052	–		0.047	0.036	–		0.049	0.055	–
0.11	0.110	0.109	–		0.109	0.098	–		0.097	0.088	–		0.103	0.101	–
CSP															
0.016	0.022	0.034	0.021		0.023	0.030	0.015		0.025	0.039	0.027		0.026	0.029	0.025
0.055	0.063	0.086	0.075		0.053	0.083	0.048		0.067	0.082	0.059		0.060	0.076	0.066
0.11	0.132	0.153	0.139		0.104	0.147	0.102		0.123	0.130	0.117		0.116	0.124	0.121
USP															
0.016	0.028	0.080	0.108		0.031	0.072	0.087		0.030	0.068	0.087		0.032	0.073	0.099
0.055	0.069	0.131	0.169		0.070	0.133	0.139		0.067	0.127	0.153		0.074	0.122	0.149
0.11	0.135	0.184	0.223		0.130	0.200	0.195		0.136	0.173	0.200		0.140	0.159	0.191
10 Replicates	Norm.H_0				Exp.H_0				t3.H_0				Weib.H_0		
	τ	θ	$(\tau\theta)$		τ	θ	$(\tau\theta)$		τ	θ	$(\tau\theta)$		τ	θ	$(\tau\theta)$
ANOVA															
0.01	0.010	0.007	0.011		0.011	0.008	0.012		0.009	0.011	0.008		0.008	0.008	0.008
0.05	0.052	0.055	0.044		0.049	0.045	0.060		0.048	0.052	0.040		0.048	0.041	0.043
0.1	0.101	0.106	0.099		0.100	0.085	0.109		0.102	0.105	0.083		0.098	0.092	0.094
Rank															
0.01	0.009	0.004	–		0.018	0.007	–		0.012	0.012	–		0.015	0.008	–
0.05	0.053	0.047	–		0.058	0.043	–		0.067	0.049	–		0.050	0.048	–
0.1	0.110	0.107	–		0.110	0.085	–		0.128	0.098	–		0.094	0.096	–
CSP															
0.01	0.019	0.031	0.014		0.017	0.024	0.018		0.012	0.025	0.015		0.014	0.016	0.010
0.05	0.060	0.079	0.057		0.073	0.064	0.068		0.079	0.083	0.057		0.061	0.070	0.056
0.1	0.109	0.129	0.124		0.122	0.125	0.122		0.133	0.129	0.106		0.111	0.118	0.112
USP															
0.01	0.031	0.071	0.066		0.032	0.059	0.076		0.036	0.076	0.064		0.023	0.059	0.068
0.05	0.082	0.138	0.152		0.083	0.116	0.154		0.090	0.140	0.136		0.081	0.123	0.138
0.1	0.127	0.190	0.221		0.132	0.157	0.214		0.154	0.183	0.181		0.123	0.168	0.194

Table B.1.: Tests under H_0 for the comparison of statistical power in a 3×5 design.

the four distribution cases defined in the previous paragraph.

We consider again the same layout with $r = 3, 5, 10$ replicates. For each scenario and each different underlying distribution of errors, we simulated again 1000 independent data sets. With $r = 5$, the CSP test is exact while in all other cases a CMC procedure with sample size 1000 is used. The Table B.3 presents the results for the power β at different nominal levels of α.

Comments. As expected, the statistical power increases significantly with the increase of replicates in all the methods. The power of USP is higher than the one of ANOVA, while CSP are slightly inferior. For all tests with equal number of replicates the power is higher if the distributions are non-normal, while it decreases with increasing number of treatments (compare results of factor τ with results of θ). As far as interaction is concerned, USP are the most powerful but we saw in Table B.1 that their type I error rate is not maintained at the nominal level; CSP, instead, behave substantially similarly to ANOVA. The main observation is, however, that rank-based tests are inferior to parametric tests only when the assumption of normality is met, while they are superior with other distributions, where the type I error rate is still maintained.

ij	11	12	13	14	15	21	22	23	24	25	31	32	33	34	35	Effect
τ	-1	-0.5	0	0.5	1	-1	-0.5	0	0.5	1	-1	-0.5	0	0.5	1	0.5
θ	-1	-1	-1	-1	-1	0	0	0	0	0	1	1	1	1	1	0.5
$(\tau\theta)$	1	0.5	0	-0.5	-1	0	0	0	0	0	-1	-0.5	0	0.5	1	0.3

Table B.2.: The coefficients of the main effects and interactions of the levels of the two factors τ and θ are listed as rows in the table. Each interaction coefficient $(\tau\theta)$ can be obtained by multiplying the corresponding τ and θ main-effect coefficients. The last column gives the entity of the effect which multiplies the coefficient. For example, $\tau_1 + \theta_1 + (\tau\theta)_{11} = -1 \cdot 0.5 - 1 \cdot 0.5 + 1 \cdot 0.3$.

B.3. A simulation study for all-pairwise comparisons

With respect to the same simulated data of the previous section we analyse the behaviour of the procedures for simultaneous confidence intervals in all-pairwise comparisons. We consider the following procedures for simultaneous confidence intervals.

Tukey's HSD given by Formula 3.15.

Protected Friedman's rank-based for comparison of average ranks given by formula 3.19 with the requirement to test first the global null hypothesis through the Friedman general test.

CSP joint comparisons with Bonferroni's adjustment computed through the function Compute_all-pairwise_MSD of Algorithm 3.6 and yielding confidence intervals of equal width valid at the Bonferroni's α_{PC} level on all comparisons.

CSP joint comparisons without adjustment computed through the function Compute_-all-pairwise_MSD of Algorithm 3.6 and yielding confidence intervals of equal width valid at the α_{FW} on all comparisons.

CSP separated comparisons with Bonferroni's adjustment computed separately through the function Compute-pairwise_MSD of algorithm 3.6 at the Bonferroni's α_{PC} yielding different confidence intervals for each pairwise comparison.

In addition to these procedures, permutation procedures are also run with USP instead of CSP.

The results of the test procedures for all pairwise comparisons that return simultaneous confidence intervals of equal width can be represented by horizontal error bars, while procedures with confidence intervals of different width for each pairwise comparison require the diagonal graphical representation. In Figure B.3 we report the visualisation of all the procedures considered. All tests refer to the same simulated design in a 3×5 layout generated under H1 using the previously defined design matrix.

In order to understand which is the procedure that behaves best we analyse the family-wise error rate and the power of the test.

B.3.1. Family-wise type I error rate

We report in Table B.4 the results of the family-wise type I error rate, that is, the probability of finding that at least one single null hypothesis $H_{0ij} : \mu_i = \mu_j$ is rejected when data are simulated under the global null hypothesis $H_0 : \mu_1 = \cdots = \mu_k$.

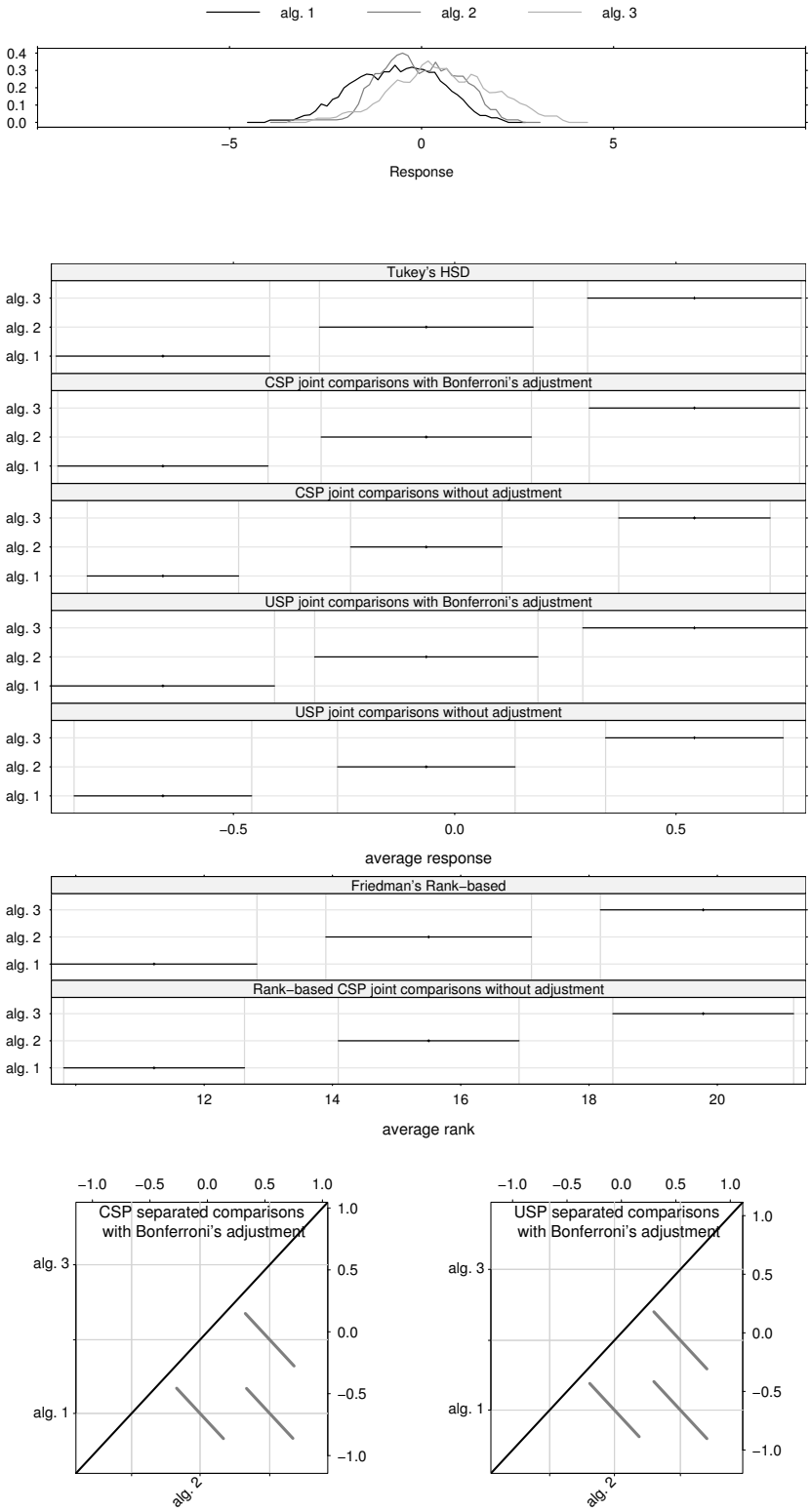


Figure B.3.: Visual comparison of different methods for simultaneous confidence intervals. The observed distributions are reported in the upper plot. For details on the graphical representation of confidence intervals see Section 3.6.2. In the diagonal representation bars in grey indicate the presence of statistical significance. The case considered uses 10 replicates.

3 Replicates	Norm.H_1			Exp.H_1			t3.H_1			Weib.H_1		
	τ	θ	$(\tau\theta)$	τ	θ	$(\tau\theta)$	τ	θ	$(\tau\theta)$	τ	θ	$(\tau\theta)$
ANOVA												
0.01	0.364	0.160	0.019	0.415	0.206	0.015	0.487	0.229	0.024	0.430	0.185	0.028
0.05	0.634	0.377	0.084	0.656	0.405	0.076	0.703	0.467	0.098	0.668	0.404	0.090
0.1	0.749	0.529	0.166	0.762	0.539	0.144	0.791	0.608	0.167	0.777	0.550	0.154
Rank												
0.01	0.260	0.109	–	0.523	0.250	–	0.519	0.238	–	0.447	0.193	–
0.05	0.522	0.306	–	0.777	0.523	–	0.789	0.503	–	0.722	0.450	–
0.1	0.653	0.452	–	0.877	0.662	–	0.868	0.669	–	0.837	0.613	–
Perm.main.same.usp												
0.01	0.454	0.425	0.148	0.518	0.464	0.133	0.585	0.533	0.175	0.525	0.471	0.145
0.05	0.672	0.591	0.257	0.696	0.599	0.235	0.732	0.674	0.272	0.710	0.617	0.250
0.1	0.765	0.663	0.322	0.779	0.676	0.300	0.811	0.740	0.328	0.796	0.686	0.313
5 Replicates	Norm.H_1			Exp.H_1			t3.H_1			Weib.H_1		
	τ	θ	$(\tau\theta)$	τ	θ	$(\tau\theta)$	τ	θ	$(\tau\theta)$	τ	θ	$(\tau\theta)$
ANOVA												
0.016	0.730	0.476	0.065	0.769	0.490	0.052	0.819	0.554	0.047	0.785	0.506	0.056
0.055	0.872	0.655	0.148	0.897	0.686	0.130	0.907	0.725	0.151	0.904	0.689	0.149
0.11	0.935	0.763	0.236	0.948	0.788	0.235	0.938	0.821	0.266	0.954	0.807	0.239
Rank												
0.016	0.655	0.370	0	0.920	0.689	0	0.911	0.670	0	0.853	0.588	0
0.055	0.818	0.582	0	0.972	0.852	0	0.975	0.822	0	0.946	0.790	0
0.11	0.892	0.723	0	0.988	0.924	0	0.986	0.895	0	0.972	0.875	0
Perm.main.same.csp												
0.016	0.590	0.396	0.068	0.594	0.441	0.075	0.668	0.491	0.079	0.610	0.421	0.075
0.055	0.804	0.616	0.156	0.819	0.666	0.144	0.853	0.696	0.174	0.835	0.668	0.151
0.11	0.902	0.754	0.245	0.917	0.790	0.258	0.920	0.811	0.292	0.921	0.785	0.264
Perm.main.same.usp												
0.016	0.817	0.713	0.211	0.842	0.758	0.219	0.888	0.789	0.256	0.866	0.764	0.221
0.055	0.901	0.799	0.290	0.927	0.826	0.312	0.931	0.852	0.358	0.938	0.841	0.305
0.11	0.948	0.852	0.360	0.959	0.880	0.398	0.948	0.886	0.428	0.961	0.875	0.377
10 Replicates	Norm.H_1			Exp.H_1			t3.H_1			Weib.H_1		
	τ	θ	$(\tau\theta)$	τ	θ	$(\tau\theta)$	τ	θ	$(\tau\theta)$	τ	θ	$(\tau\theta)$
ANOVA												
0.01	0.978	0.822	0.094	0.971	0.824	0.099	0.966	0.843	0.122	0.973	0.849	0.100
0.05	0.994	0.942	0.244	0.989	0.935	0.260	0.987	0.934	0.299	0.995	0.944	0.261
0.1	0.999	0.976	0.380	0.994	0.960	0.376	0.991	0.967	0.431	0.998	0.971	0.380
Rank												
0.01	0.956	0.738	0	0.998	0.977	0	1	0.965	0	0.997	0.918	0
0.05	0.990	0.915	0	1.000	0.995	0	1	0.994	0	1.000	0.985	0
0.1	0.997	0.953	0	1.000	1.000	0	1	0.998	0	1.000	0.994	0
Perm.main.same.csp												
0.01	0.952	0.815	0.108	0.948	0.829	0.143	0.955	0.839	0.168	0.953	0.822	0.121
0.05	0.990	0.947	0.273	0.985	0.938	0.297	0.983	0.935	0.354	0.996	0.946	0.307
0.1	0.997	0.971	0.403	0.994	0.967	0.409	0.990	0.963	0.463	0.998	0.973	0.410
Perm.main.same.usp												
0.01	0.990	0.956	0.311	0.983	0.946	0.322	0.985	0.957	0.378	0.992	0.959	0.310
0.05	0.998	0.983	0.447	0.993	0.973	0.468	0.990	0.975	0.515	0.997	0.980	0.474
0.1	1.000	0.987	0.537	0.997	0.979	0.561	0.995	0.983	0.595	1.000	0.989	0.542

Table B.3.: Tests under H_1 for the comparison of statistical power in a 3×5 design.

Results refer to a nominal α_{FW} of 0.05 and are obtained by 1000 simulated data from the 3×5 design with $r = \{3, 5, 10\}$.

Comments. The first observation concerns the parametric Tukey's HSD method that remains sufficiently robust under all different underlying distributions. Friedman's rank-based method is also sufficiently robust, although in this case we must add that the result observed is mainly due to the protection on the global hypothesis. Experiments without protection showed that the family-wise error rate of this method is quite high. As far as permutation methods are concerned, we should distinguish between CSP and USP in relation to the number of replicates.

In the case of CSP with only 5 replicates the level of significance per comparison adjusted by Bonferroni's rule becomes smaller than the minimal observable α value, therefore in those cases we do not report results. The error rate remains however slightly higher for CSP with respect to USP even for the factor τ and therefore USP should still be preferred even with 5 replicates in pairwise-comparisons.

The use of Bonferroni's adjustment in the joint comparisons procedures makes the tests more conservative. However, for $r = 10$ the type I error for the procedure without adjustment becomes too high, above all, for the factor θ that has 5 levels. An adjustment

3 Replicates	Norm.H_0		Exp.H_0		t3.H_0		Weib.H_0	
	τ	θ	τ	θ	τ	θ	τ	θ
Tukey's HSD	0.052	0.061	0.047	0.048	0.057	0.055	0.045	0.039
Protected Friedman's rank-based	0.042	0.048	0.041	0.047	0.058	0.047	0.050	0.042
USP joint comparisons with Bonferroni's adjustment	0.004	0.003	0.005	0.002	0.004	0.004	0.001	0.002
USP joint comparisons without adjustment	0.017	0.024	0.016	0.007	0.019	0.020	0.017	0.014
USP separated comparisons with Bonferroni's adjustment	0.010	0.063	0.020	0.064	0.011	0.044	0.019	0.055
5 Replicates	Norm.H_0		Exp.H_0		t3.H_0		Weib.H_0	
	τ	θ	τ	θ	τ	θ	τ	θ
Tukey's HSD	0.049	0.048	0.046	0.025	0.038	0.052	0.054	0.044
Protected Friedman's rank-based	0.055	0.050	0.046	0.029	0.047	0.053	0.051	0.055
CSP joint comparisons with Bonferroni's adjustment	0.018	–	0.014	–	0.003	–	0.013	–
CSP joint comparisons without adjustment	0.050	0.031	0.044	0.014	0.036	0.032	0.046	0.023
CSP separated comparisons with Bonferroni's adjustment	0.076	–	0.078	–	0.068	–	0.077	–
USP joint comparisons with Bonferroni's adjustment	0.012	0.006	0.007	0.001	0.004	0.003	0.004	0.004
USP joint comparisons without adjustment	0.040	0.065	0.033	0.016	0.031	0.039	0.035	0.035
USP separated comparisons with Bonferroni's adjustment	0.020	0.035	0.021	0.032	0.017	0.041	0.026	0.036
10 Replicates	Norm.H_0		Exp.H_0		t3.H_0		Weib.H_0	
	τ	θ	τ	θ	τ	θ	τ	θ
Tukey's HSD	0.045	0.053	0.043	0.049	0.040	0.055	0.049	0.044
Protected Friedman's rank-based	0.042	0.053	0.039	0.056	0.035	0.056	0.039	0.065
CSP joint comparisons with Bonferroni's adjustment	0.028	0.007	0.018	0.002	0.018	0.005	0.023	0.004
CSP joint comparisons without adjustment	0.084	0.116	0.075	0.100	0.056	0.119	0.088	0.127
CSP separated comparisons with Bonferroni's adjustment	0.061	0.065	0.046	0.079	0.046	0.062	0.057	0.079
USP joint comparisons with Bonferroni's adjustment	0.033	0.025	0.022	0.011	0.028	0.021	0.036	0.016
USP joint comparisons without adjustment	0.104	0.193	0.084	0.152	0.085	0.174	0.095	0.172
USP separated comparisons with Bonferroni's adjustment	0.056	0.089	0.061	0.096	0.051	0.098	0.058	0.098

Table B.4.: Family-wise error rate at the α level of 0.05 in a 3×5 design under H_0.

seems therefore needed, although Bonferroni's one is conservative. The separated comparisons procedure behaves the best among permutation tests although it also worsens considerably on the factor θ and $r = 10$.

B.3.2. Comparison of power

We provide two tables for understanding the statistical power of the all-pairwise comparison procedures. Both tables are based on the simulation setting of the previous paragraph but with active effects determined as indicated by the design matrix of Table B.2. For each procedure we report in Table B.5 the rate of success of in determining that *at least one* comparison is significant while in Table B.6 we report the rate of success in determining that *each* individual comparison involving two algorithms is significant. In both cases the nominal α_{FW} value is 0.05.

Comments. We point out the following observations.

- In general the two tables lead to a consistent assessment of the procedures' behaviour.
- For all procedures, the power increases with increasing number of replicates, decreases with increasing number of treatments (see differences between τ and θ), and is maintained on an equal level with different distributions.
- The rank-based procedure is more powerful than Tukey's method if the distributions are not normal and it is in absolute the most powerful procedure, especially with increasing number of algorithms.
- USP are always more powerful than CSP. CSP are to be preferred at $r = 10$, given the smaller type I error rate while in the other cases USP behave better.

3 Replicates	**Norm.** H_0		**Exp.** H_0		**t3.** H_0		**Weib.** H_0	
	τ	θ	τ	θ	τ	θ	τ	θ
Tukey's HSD	0.650	0.353	0.657	0.358	0.767	0.438	0.688	0.399
Protected Friedman's rank-based	0.559	0.289	0.773	0.502	0.813	0.531	0.718	0.446
USP joint comparisons with Bonferroni's adjustment	0.285	0.051	0.303	0.055	0.443	0.102	0.312	0.071
USP joint comparisons without adjustment	0.492	0.221	0.466	0.177	0.627	0.280	0.484	0.242
USP separated comparisons with Bonferroni's adjustment	0.427	0.305	0.484	0.431	0.628	0.508	0.483	0.401
5 Replicates	**Norm.** H_0		**Exp.** H_0		**t3.** H_0		**Weib.** H_0	
	τ	θ	τ	θ	τ	θ	τ	θ
Tukey's HSD	0.894	0.616	0.866	0.631	0.882	0.693	0.891	0.626
Protected Friedman's rank-based	0.843	0.569	0.963	0.841	0.968	0.814	0.949	0.748
CSP joint comparisons with Bonferroni's adjustment	0.412	–	0.450	–	0.536	–	0.470	–
CSP joint comparisons without adjustment	0.725	0.361	0.705	0.388	0.752	0.437	0.721	0.380
CSP separated comparisons with Bonferroni's adjustment	0.648	–	0.685	–	0.757	–	0.700	–
USP joint comparisons with Bonferroni's adjustment	0.717	0.241	0.715	0.242	0.770	0.332	0.731	0.258
USP joint comparisons without adjustment	0.879	0.651	0.830	0.600	0.860	0.654	0.864	0.621
USP separated comparisons with Bonferroni's adjustment	0.801	0.539	0.805	0.652	0.856	0.763	0.815	0.605
10 Replicates	**Norm.** H_0		**Exp.** H_0		**t3.** H_0		**Weib.** H_0	
	τ	θ	τ	θ	τ	θ	τ	θ
Tukey's HSD	0.999	0.941	0.991	0.922	0.986	0.925	0.999	0.929
Protected Friedman's rank-based	0.993	0.924	1.000	0.997	1.000	0.995	1.000	0.975
CSP joint comparisons with Bonferroni's adjustment	0.985	0.608	0.970	0.634	0.972	0.662	0.990	0.612
CSP joint comparisons without adjustment	0.999	0.964	0.990	0.943	0.989	0.940	0.999	0.949
CSP separated comparisons with Bonferroni's adjustment	0.998	0.912	0.989	0.938	0.989	0.946	0.996	0.915
USP joint comparisons with Bonferroni's adjustment	0.999	0.867	0.987	0.848	0.987	0.869	0.996	0.849
USP joint comparisons without adjustment	1.000	0.986	0.995	0.967	0.997	0.972	1.000	0.974
USP separated comparisons with Bonferroni's adjustment	1.000	0.961	0.996	0.975	0.994	0.978	0.999	0.960

Table B.5.: The rate with which *at least one* algorithm is recognised significantly different at an α level of 0.05 in a 3×5 design under H_0.

3 Replicates	**Norm.** H_1		**Exp.** H_1		**t3.** H_1		**Weib.** H_1	
	τ	θ	τ	θ	τ	θ	τ	θ
Tukey's HSD	0.152	0.031	0.161	0.034	0.195	0.043	0.170	0.040
Protected Friedman's rank-based	0.163	0.058	0.248	0.109	0.260	0.111	0.216	0.092
USP joint comparisons with Bonferroni's adjustment	0.054	0.003	0.060	0.004	0.088	0.007	0.063	0.005
USP joint comparisons without adjustment	0.102	0.017	0.100	0.014	0.143	0.024	0.106	0.020
USP separated comparisons with Bonferroni's adjustment	0.085	0.024	0.103	0.040	0.138	0.049	0.105	0.037
5 Replicates	**Norm.** H_1		**Exp.** H_1		**t3.** H_1		**Weib.** H_1	
	τ	θ	τ	θ	τ	θ	τ	θ
Tukey's HSD	0.237	0.063	0.243	0.066	0.258	0.080	0.240	0.069
Protected Friedman's rank-based	0.262	0.113	0.350	0.193	0.352	0.186	0.326	0.165
CSP joint comparisons with Bonferroni's adjustment	0.090	–	0.099	–	0.126	–	0.102	–
CSP joint comparisons without adjustment	0.182	0.034	0.180	0.038	0.202	0.048	0.183	0.040
CSP separated comparisons with Bonferroni's adjustment	0.160	–	0.171	–	0.203	–	0.171	–
USP joint comparisons with Bonferroni's adjustment	0.157	0.017	0.164	0.017	0.184	0.027	0.159	0.017
USP joint comparisons without adjustment	0.222	0.070	0.226	0.062	0.240	0.077	0.225	0.069
USP separated comparisons with Bonferroni's adjustment	0.188	0.050	0.205	0.068	0.228	0.085	0.197	0.064
10 Replicates	**Norm.** H_1		**Exp.** H_1		**t3.** H_1		**Weib.** H_1	
	τ	θ	τ	θ	τ	θ	τ	θ
Tukey's HSD	0.352	0.139	0.350	0.142	0.369	0.155	0.359	0.145
Protected Friedman's rank-based	0.379	0.214	0.458	0.299	0.459	0.288	0.440	0.269
CSP joint comparisons with Bonferroni's adjustment	0.300	0.055	0.300	0.059	0.319	0.071	0.310	0.060
CSP joint comparisons without adjustment	0.368	0.176	0.370	0.175	0.385	0.181	0.378	0.178
CSP separated comparisons with Bonferroni's adjustment	0.337	0.131	0.341	0.143	0.368	0.163	0.348	0.142
USP joint comparisons with Bonferroni's adjustment	0.338	0.108	0.337	0.106	0.358	0.121	0.348	0.109
USP joint comparisons without adjustment	0.391	0.209	0.393	0.201	0.407	0.208	0.400	0.205
USP separated comparisons with Bonferroni's adjustment	0.362	0.161	0.367	0.177	0.390	0.192	0.373	0.174

Table B.6.: The probability of recognising *any* individual comparison as significant at an α level of 0.05 in a 3×5 design under H_1.

– The *separated comparisons with Bonferroni's adjustment* and the *joint comparisons without adjustment* have similar power, hence, given the result on the type I error the first has to be preferred.

– The *joint comparisons with Bonferroni's adjustment* exhibit almost the half of the power of all other methods, and only with USP and $r = 10$ it becomes competitive.

We may conclude that the best behaviour is exhibited by rank-based tests but USP and CSP tests with separated comparisons and Bonferroni's adjustment are also very competitive.

B.4. Discussion

We presented a comparative study of permutation, parametric, and rank-based tests. The advantage of permutation tests is that they do not require the underlying data to be normally distributed. However, we observed that their type I error is often higher than the one of parametric tests that, in constrast, remain particularly robust on data from different theoretical distributions. The robustness of parametric tests is known in Statistics. Here we extended it to two asymmetric distributions, namely the exponential distribution and the Weibull distributions, which are particularly likely to appear for data from stochastic optimisers. Although the results presented may depend on the particular parametrisation adopted for the generation of these distributions, we believe that they may be highly indicative.

As far as permutation tests are concerned, we observed that for general hypothesis testing, CSP behave well already with $r = 5$ and the use of USP in this case could be confined to cases with $r < 5$, although their type I error rate results higher than the nominal one. For $r \geq 5$, CSP behave better and the type I error rate is only slightly higher than the nominal while the power is in general comparable with the one of parametric tests. Turning our attention to procedures for all-pairwise comparisons we have the following conclusions: (i) in general the behaviour of permutation tests indicates that they require some further efforts for calibration, as they errors do not yet match well those declared (ii) the adoption of USP is necessary also with 5 replicates while CSP is applicable at 10, (iii) the procedure *separated comparisons with Bonferroni's adjustment* is the one that exhibits the best behaviour, (iv) the procedure *joint comparisons without Bonferroni's adjustment* exhibits a type I error rate which is too high and the procedure *joint comparisons with Bonferroni's adjustment* has to be preferred although very conservative.

We deem that the general outcome of this analysis is that rank-based tests are currently the best choice for comparing algorithms, as they are the most powerful and maintain the nominal type I error. In addition, we saw clearly that the increase of the number of algorithms tested requires as counteraction an increase of the number of replicates to maintain the statistical power at a constant level.

The analysis is however far away from being exhaustive. It would be interesting to study the behaviour of the tests in relation with the increase of the size of the design. A simulation study in which data are collected from real results of algorithms rather than randomly generated from theoretical distributions would be closer to the scope of combinatorial optimisation. For example, a comparison under the null hypothesis could easily be obtained by generating data with a unique algorithm or by shuffling data obtained from different algorithms. The simulation under the alternative hypothesis appears, instead, more problematic given that we would need to know *a priori* if the algorithms are actually different. In addition, a similar analysis should be developed for the scenario *"one single run on various instances"*. However, the most relevant (and urgent) extension of the analysis, would be the study of the robustness of these or other tests to distributions that may actually have characteristics like non-homoschedasticity and multi-modality.

Appendix C.

Numerical Results on the Benchmark Instances for Graph Colouring Problems

In the main body of this thesis we gave preference to the graphical representation of results. In this appendix, we present the numerical results on which the graphics are based.

C.1. Graph colouring

C.1.1. Instance statistics

We present some descriptive statistics for all the 125 instances from the DIMACS repository included in our analysis.[1] Instances are divided in two Tables, Table C.1 for the easy instances and Table C.2 for the hard instances. For details on this classification, we refer to Section 4.5. An asterisc indicates that the graph is actually disconnected.

For each instance we report the number of vertices $|V|$; the edge density $\rho(G)$; the average vertex degree $d(G)$; its standard deviation s_d; the percentage % of the vertices that is removed by the preprocessing stage described in Section 4.5; the chromatic number $\chi(G)$ of G, when known; and the maximum clique size $\hat{\omega}(G)$ found by heuristic algorithms. This latter value is obtained by a modification of the "semi-exhaustive greedy" scheme for finding large independent sets used in XRLFJohnson et al. (1991)[2] applied on the complement graph. In the few cases where the output of this algorithm was worse than that the output of the reactive tabu search algorithm of Battiti and Protasi (2001), we report the latter value between parenthesis. The computation time is relative to the algorithm of Johnson et al. (1991).

Finally, the median number of colours produced in 10 runs by each of the three construction heuristics is also given. Furthermore, only for RLF, we also report the median computation time (see Section 4.4 for details on the machine used).

[1] The number of graph colouring instances at the COLOR02/03/04 web-page (`http://mat.gsia.cmu.edu/COLOR04/`) is 119. We added the Flat graphs from the DIMACS web site `http://mat.gsia.cmu.edu/COLOR/instances.html`.

[2] The following set of parameters is assumed: `TRIALNUM` $= 4$, `ITERATIONS` $= 1000$, `CANDNUM` $= |V|/4.5$ and `SETLIM` $= 100$ for $\rho(G) \leq 0.5$, 50 for $0.5 < \rho(G) \leq 0.5$ and 30 otherwise.

Instance	Statistics						Reduc.	$\chi(G)$	Clique		ROS	DSATUR	RLF	
	$\|V\|$	$\rho(G)$	$d(G)$	s_d	$\delta(G)$	$\Delta(G)$	%		$\hat{\omega}(G)$	sec.	# col.	# col.	# col.	sec.
2-Insertions_3	37	0.1081	3.9	1	3	9	0	4	2	0.15	4	4	4	0
3-Insertions_3	56	0.0714	3.9	1	3	11	0	4	2	0.33	4	4	4	0
DSJR500.1	500	0.0285	14.2	4.4	4	25	100	–	12	1.79	15	13	13	0
anna	138	0.0522	7.1	10.4	1	71	100	–	11	0.17	11	11	11	0
david	87	0.1085	9.3	10.5	1	82	100	–	11	0.8	12	11	11	0
fpsol2.i.1*	496	0.0949	47	63.1	0	252	100	65	65	5.26	65	65	65	0
fpsol2.i.2*	451	0.0856	38.5	63.1	0	346	81	30	30	2.22	31	30	30	0
fpsol2.i.3*	425	0.0964	40.9	64.3	0	346	80	30	30	2.17	30	30	30	0
games120	120	0.0894	10.6	1.2	7	13	100	9	9	0.12	9	9	9	0
homer*	561	0.0104	5.8	10.2	0	99	100	13	13	2.18	14	13	13	0
huck*	74	0.1114	8.1	7.3	1	53	100	11	11	0.58	11	11	11	0
inithx.i.1*	864	0.0502	43.3	72.7	0	502	89	54	54	9.57	54	54	54	0.1
inithx.i.2*	645	0.0673	43.3	73.7	0	541	81	31	31	4.61	31	31	31	0
inithx.i.3*	621	0.0726	45	74.8	0	542	80	31	31	4.45	31	31	31	0
jean*	80	0.0804	6.3	6	0	36	100	10	10	0.66	10	10	10	0
le450_25a	450	0.0818	36.7	23	2	128	41	25	25	1.74	28	25	25	0
le450_25b	450	0.0818	36.7	21.1	2	111	35	25	25	1.73	28	25	25	0
le450_5c	450	0.097	43.6	6.4	27	66	0	5	5	1.77	17	7	5	0
miles1000	128	0.3957	50.2	20	13	86	100	42	42	0.55	45	42	42	0
miles1500	128	0.6395	81.2	23.3	28	106	100	73	73	2.64	74	73	73	0
miles250*	128	0.0476	6	3.6	0	16	100	8	8	0.13	9	8	8	0
miles500	128	0.1439	18.3	9.7	3	38	100	20	20	0.2	22	20	20	0
miles750	128	0.26	33	15.2	6	64	100	31	31	0.29	34	31	31	0
mug100_1	100	0.0335	3.3	0.5	3	4	0	4	3	0.97	4	4	4	0
mug100_25	100	0.0335	3.3	0.5	3	4	0	4	3	0.98	4	4	4	0
mug88_1	88	0.0381	3.3	0.5	3	4	0	4	3	0.76	4	4	4	0
mug88_25	88	0.0381	3.3	0.5	3	4	0	4	3	0.76	4	4	4	0
mulsol.i.1*	197	0.2033	39.8	31	0	121	100	49	49	0.57	49	49	49	0
mulsol.i.2*	188	0.221	41.3	31.1	0	156	62	31	31	0.55	31	31	31	0
mulsol.i.3*	184	0.2326	42.6	31	0	157	61	31	31	0.57	31	31	31	0
mulsol.i.4*	185	0.2318	42.7	31.2	0	158	61	31	31	0.57	31	31	31	0
mulsol.i.5*	186	0.2309	42.7	31.4	0	159	61	31	31	0.6	31	31	31	0
myciel3	11	0.3636	3.6	0.6	3	5	0	4	2	0.02	4	4	4	0
myciel4	23	0.2806	6.2	1.9	4	11	0	5	2	0.07	5	5	5	0
myciel5	47	0.2183	10	4.4	5	23	0	6	2	0.28	6	6	6	0
myciel6	95	0.1691	15.9	8.8	6	47	0	7	2	1.09	7	7	7	0
myciel7	191	0.1301	24.7	16.6	7	95	0	8	2	0.36	8.5	8	8	0
qg.order30	900	0.0645	58	0	58	58	0	30	30	6.13	34	31	30	0.1
qg.order40	1600	0.0488	78	0	78	78	0	40	40	19.59	44	42	40	0.3
qg.order60	3600	0.0328	118	0	118	118	0	60	60	110.07	65	62	61	1.5
queen5_5	25	0.5333	12.8	1.1	12	16	0	5	5	0.1	7.5	5	5	0
wap05a	905	0.1053	95.2	47.1	9	228	27	–	50	8.46	55	50	50	0.2
zeroin.i.1*	211	0.1851	38.9	38.7	0	111	100	49	49	0.75	49	49	49	0
zeroin.i.2*	211	0.1598	33.6	36.3	0	140	100	30	30	0.7	31	30	30	0
zeroin.i.3*	206	0.1677	34.4	36.4	0	140	100	30	30	0.69	31	30	30	0

Table C.1.: Characteristics of the instances classified as "easy" and results of the construction heuristics.

Name instance	Statistics						Reduc.	$\chi(G)$	Clique		ROS	DSATUR	RLF	
	$\|V\|$	$\rho(G)$	$d(G)$	s_d	$\delta(G)$	$\Delta(G)$	%		$\hat{\omega}(G)$	sec.	# col.	# col.	# col.	sec.
1-FullIns_3	30	0.2299	6.7	2.1	4	11	50	–	3	0.11	5	4	4	0
1-FullIns_4	93	0.1386	12.8	5.5	6	32	62	–	3	0.99	6	5	5	0
1-FullIns_5	282	0.082	23	13.2	8	95	73	–	3	0.65	7	6	6	0
2-FullIns_3	52	0.1516	7.7	3	4	15	83	–	4	0.31	5.5	5	5	0
2-FullIns_4	212	0.0725	15.3	8.2	6	55	81	–	4	0.36	8	6	6	0
2-FullIns_5	852	0.0337	28.6	20.4	8	215	90	–	4	5.17	9.5	7	7	0
3-FullIns_3	80	0.1095	8.6	4	4	19	86	–	5	0.68	6.5	6	6	0
3-FullIns_4	405	0.0431	17.4	10.8	6	84	89	–	5	1.22	9	7	7	0
3-FullIns_5	2030	0.0164	33.3	28.1	8	409	95	–	5	30.3	11	8	8	0
4-FullIns_3	114	0.084	9.5	5	4	23	89	–	6	0.11	7.5	7	7	0
4-FullIns_4	690	0.028	19.3	13.5	6	119	95	–	6	3.36	9.5	8	8	0
4-FullIns_5	4146	0.009	37.3	36.2	8	695	97	–	6	144.31	12.5	9	9	0.1
5-FullIns_3	154	0.0672	10.3	5.9	4	27	90	–	7	0.2	9	8	8	0
5-FullIns_4	1085	0.0194	21	16.1	6	160	96	–	7	8.14	11	9	9	0
1-Insertions_4	67	0.1049	6.9	2.7	4	22	0	4	2	0.49	5	5	5	0
1-Insertions_5	202	0.0604	12.1	6.7	5	67	0	–	2	0.31	6.5	6	6	0
1-Insertions_6	607	0.0345	20.9	14.6	6	202	0	–	2	2.62	8	7	7	0
2-Insertions_4	149	0.0491	7.3	3.2	4	37	0	4	2	0.17	5	5	5	0
2-Insertions_5	597	0.0221	13.2	8.5	5	149	0	–	2	2.49	7	6	6	0
3-Insertions_4	281	0.0266	7.4	3.7	4	56	0	–	2	0.57	6	5	5	0
3-Insertions_5	1406	0.0098	13.8	10.1	5	281	0	–	2	13.65	8	6	6	0
4-Insertions_3	79	0.0506	3.9	1.1	3	13	0	3	2	0.63	4	4	4	0
4-Insertions_4	475	0.0159	7.6	4	4	79	0	–	2	1.57	6	5	5	0
DSJC125.1	125	0.095	11.8	3.3	5	23	0	–	4	0.14	8	6	6	0
DSJC125.5	125	0.5021	62.3	5.3	51	75	0	–	10	0.52	25	21.5	21	0
DSJC125.9	125	0.8982	111.4	3.2	103	120	0	–	34	1.78	56	52	49	0
DSJC250.1	250	0.1034	25.7	5.1	13	38	0	–	4	0.55	13	10	10	0
DSJC250.5	250	0.5034	125.3	7.8	101	147	0	–	12	1.64	42	37	35	0
DSJC250.9	250	0.8963	223.2	4.6	207	234	0	–	43	6.75	96	89.5	85	0.1
DSJC500.1	500	0.0999	49.8	6.7	34	68	0	–	5	2.17	20	16	15	0
DSJC500.5	500	0.502	250.5	11	220	286	0	–	13	6.06	73	65	61	0.3
DSJC500.9	500	0.9013	449.7	6.4	430	471	0	–	54 (56)	29.4	178.5	165.5	157	1.2
DSJC1000.1	1000	0.0994	99.3	9.5	68	127	0	–	6	8.52	31	27	24	0.1
DSJC1000.5	1000	0.5002	499.7	15.8	447	551	0	–	15	24.34	128	116	108	1.6
DSJC1000.9	1000	0.8998	898.9	9.6	870	924	0	–	63 (67)	116.14	320.5	302.5	284.5	11.9
DSJR500.1c	500	0.9721	485.1	4.2	473	497	42	–	81 (83)	77.85	107.5	89	93	0.8
DSJR500.5	500	0.4718	235.4	64.6	103	388	3	–	122	56.75	146	130	131	0.5
abb313GPIA*	1557	0.0384	59.8	26.2	0	151	51	–	8	18.99	16	11	11	0.1
ash331GPIA	662	0.0191	12.6	2.8	1	23	0	–	3	3.05	6	5	4.5	0
ash608GPIA	1216	0.0106	12.9	2.7	1	20	0	–	3	10.25	7	5	5	0
ash958GPIA	1916	0.0068	13.1	2.8	1	24	0	–	3	27.05	7	5.5	5	0
will199GPIA	701	0.0276	19.3	7.2	2	38	6	–	6	3.47	11	7	7	0
flat300_20_0	300	0.4766	142.5	6.3	127	160	0	–	11	3.34	46	41	38	0
flat300_26_0	300	0.4823	144.2	6.2	123	158	0	–	11	3.52	46.5	42	39	0.1
flat300_28_0	300	0.4837	144.6	6.4	130	162	0	–	12	3.6	47	42	39	0
flat1000_50_0	1000	0.4905	490	9.7	459	520	0	–	14	24.48	124	115	106	1.6
flat1000_60_0	1000	0.4922	491.7	11.4	457	524	0	–	15	24.8	124	115	106.5	1.6
flat1000_76_0	1000	0.4939	493.4	12.3	455	532	0	–	14 (15)	25.13	125	115.5	108	1.7
latin_square_10	900	0.7597	683	0	683	683	0	–	90	92.99	152.5	132.5	133.5	2.5
qg.order100	10000	0.0198	198	0	198	198	0	100	100	965.22	107	103	101	18.5
le450_5a	450	0.0566	25.4	5.4	13	42	0	5	5	1.55	13	10	7	0
le450_5b	450	0.0568	25.5	5.1	12	42	0	5	5	1.55	13	10	7	0
le450_5d	450	0.0966	43.4	6.6	29	68	0	5	5	1.77	17	11	5.5	0
le450_15a	450	0.0809	36.3	16.8	2	99	10	15	15	1.7	22	16	16.5	0
le450_15b	450	0.0809	36.3	16.4	1	94	9	15	15	1.69	22	16	16	0
le450_15c	450	0.1651	74.1	21.2	18	139	0	15	15	2.33	31	24	23	0
le450_15d	450	0.1658	74.4	21.2	18	138	0	15	15	2.34	30	24	23	0
le450_25c	450	0.1717	77.1	29.2	7	179	3	25	25	2.41	36	29	28	0
le450_25d	450	0.1725	77.4	28.5	11	157	4	25	25	2.42	36	29	28.5	0
queen6_6	36	0.4603	16.1	1.4	15	19	0	7	6	0.2	9	9	8	0
queen7_7	49	0.4048	19.4	1.6	18	24	0	7	7	0.35	11	10	9	0
queen8_12	96	0.3	28.5	2.3	25	32	0	12	12	1.14	15	13.5	13	0
queen8_8	64	0.3611	22.8	1.9	21	27	0	9	8	0.58	13	11	10	0
queen9_9	81	0.3259	26.1	2.1	24	32	0	10	9	0.91	14	13	11	0
queen10_10	100	0.297	29.4	2.3	27	35	0	11	10	1.35	15.5	14	13	0
queen11_11	121	0.2727	32.7	2.6	30	40	0	11	11	0.26	17	15	14	0
queen12_12	144	0.2521	36.1	2.8	33	43	0	12	12	0.34	18	16.5	15	0
queen13_13	169	0.2344	39.4	3	36	48	0	13	13	0.43	19	18	16	0
queen14_14	196	0.219	42.7	3.3	39	51	0	14	14	0.53	21	19.5	17	0
queen15_15	225	0.2056	46	3.5	42	56	0	15	15	0.66	22.5	21	18	0
queen16_16	256	0.1936	49.4	3.8	45	59	0	16	16	0.81	23	21	19	0
school1*	385	0.2583	99.2	49.9	1	282	8	–	14	1474.34	41	15	25	0
school1_nsh*	352	0.2365	83	35.1	1	232	7	–	14	385.77	38	27	25	0
wap01a	2368	0.0396	93.6	48.2	14	288	26	–	41	47.4	57	49	47	0.5
wap02a	2464	0.0368	90.7	49.7	14	294	31	40	40	51.31	55	46	44	0.5
wap03a	4730	0.0256	121.2	59.1	26	344	14	–	40	200.52	62	55	50	1.6
wap04a	5231	0.0216	112.8	63.8	26	351	23	40	40	242.19	61	48	46	1.7
wap06a	947	0.0973	92	48.8	9	230	26	40	40	8.92	54	49	44	0.2
wap07a	1809	0.0632	114.3	54.3	13	298	11	–	40	29.2	59	46	46	0.4
wap08a	1870	0.0596	111.4	56	13	308	15	40	40	30.8	58	45	45	0.5

Table C.2.: Characteristics of the instances classified as "hard" and results of the construction heuristics.

Instance	SETLIM	CANDNUM	EXACTLIM	TRIALNUM
DSJC125.5.col	126	1	75	1
DSJC250.5.col	63	50	65	160
DSJC500.5.col	63	50	70	1280
DSJC1000.5.col	63	50	70	1280
DSJC125.1.col	20	50	125	1
DSJC250.1.col	20	50	125	1000
DSJC500.1.col	20	50	100	320
DSJC1000.1.col	20	50	100	280
DSJC125.9.col	126	1	80	1
DSJC250.9.col	251	1	80	1
DSJC500.9.col	501	1	80	1280
DSJC1000.9.col	250	50	70	20
DSJR500.1c.col	501	1	300	1
DSJR500.5.col	250	5	0	10
Queens	–	50	40	5-10
School	–	35	25	5
Jac	–	50	25	5
Wap	–	50	40	10
Flat	–	50	60	100
Leighton	–	50	60	100
FullIns	–	50	30	2
Insertion	–	50	30	2
$\|V\|$=500 ρ =0.1	–	50	55	30
$\|V\|$=500 ρ =0.5	–	50	75	60
$\|V\|$=500 ρ =0.9	–	50	75	60
$\|V\|$=1000 ρ =0.1	–	50	55	30
$\|V\|$=1000 ρ =0.5	–	50	75	60
$\|V\|$=1000 ρ =0.9	–	50	75	60

Table C.3.: The parameter settings used for XRLF.

C.1.2. The parameter set used for XRLF

The algorithm XRLF of Johnson et al. (1991) required a special tuning of its parameters to work in the context of our experimental design. We report in Table C.3 the settings we selected. We refer the reader to the original paper for details on the meaning of the parameters.

C.1.3. Machine benchmark

From the web site of the challenge "Computational Series: Graph Coloring and its Generalizations" a benchmark code is available for the comparison of machines. We report here the results on our machine, a 2 GHz AMD Athlon MP 2400+ Processor with 256 KB cache and 1 GB RAM, running Linux Debian when the code is compiled with the GNU C compiler version 3.3.5. The compilation is done using the optimisation flag -O3, as used for all algorithms discussed in this thesis.

```
DFMAX(r100.5.b)
    0.00 (user)      0.00 (sys)      0.00 (real)
Best: 4 57 35 5 61 34 3 62 90

DFMAX(r200.5.b)
    0.04 (user)      0.00 (sys)      0.00 (real)
Best: 113 86 147 66 14 134 32 127 161 186 70

DFMAX(r300.5.b)
    0.37 (user)      0.00 (sys)      1.00 (real)
Best: 279 222 116 17 39 127 190 158 196 288 263 54
```

```
DFMAX(r400.5.b)
    2.26 (user)      0.00 (sys)      2.00 (real)
Best: 370 108 27 50 87 275 145 222 355 88 306 335 379

DFMAX(r500.5.b)
    8.64 (user)      0.00 (sys)      9.00 (real)
Best: 345 204 148 480 16 336 76 223 260 403 141 382 289
```

C.1.4. Detailed results

Table C.4 reports the numerical details for the experimental evaluation of the algorithms described in Section 4.11. For each instance, we report $\chi(G)$ (if known) or the best colouring found in the literature $\widehat{\chi}^{best}(G)$, and the time limit we used for our experiments (we recall that this is given by the median time needed by $\mathsf{TS}_{\mathcal{N}_1}$ to perform $10000 \times |V|$ iterations on the instance). Then, for each algorithm we report the minimum and the median number of colours found; in addition, we give the time after which the probability of attaining a solution better than the median solution quality becomes inferior to 50%. These time values are obtained from the median attainment curves illustrated in Section 4.12.1 at which a last improvement occur. Algorithms are ordered from left to right according to the rank in the aggregate results. A grey background is used to emphasise that the algorithms performed significantly better than the others on the specific instance. If more than one algorithm is with grey background, then the differences among them are not statistically signifcant. Differently to what presented in the text, the analysis follows a *"one single run on various instances"* design and is carried out by means of the Kruskall-Wallis test for the global hypothesis and the Mann-Whitney test with level of significance adjusted by the method of Holm's procedure for the all-pairwise comparisons (see Section 3.6.2).

Instance	Bench.	time	$TS_{\mathcal{N}_1}$			HEA			ILS			MC-$TS_{\mathcal{N}_1}$			GLS			Nov^+			$SA_{\mathcal{N}_6}$			TS_{VLSN}			XRLF		
	$(\chi,\widehat{\chi}^{best})$	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.	min	med	sec.
1-FullIns_3	(4,4)	1	4	4	0	4	4	0	4	4	0	4	4	0	4	4	0	4	4	0	4	4	0	4	4	0	4	4	0
1-FullIns_4	(5,5)	3	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0
1-FullIns_5	(–,6)	11	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	10.5
2-FullIns_3	(5,5)	1	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0
2-FullIns_4	(–,6)	7	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	6.4
2-FullIns_5	(–,7)	33	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	6.3
3-FullIns_3	(6,6)	2	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0
3-FullIns_4	(–,7)	13	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0.6
3-FullIns_5	(–,8)	87	8	8	0	8	8	0	8	8	0	8	8	0	8	8	0.1	8	8	0	8	8	0	8	8	0	8	8	11.8
4-FullIns_3	(7,7)	3	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0.6
4-FullIns_4	(–,8)	23	8	8	0	8	8	0	8	8	0	8	8	0	8	8	0	8	8	0	8	8	0	8	8	0	8	8	2.1
4-FullIns_5	(–,9)	188	9	9	0.2	9	9	0.2	9	9	0.2	9	9	0.2	9	9	0.2	9	9	0.2	9	9	0.1	9	9	0.2	9	9	63.2
5-FullIns_3	(8,8)	5	8	8	0	8	8	0	8	8	0	8	8	0	8	8	0	8	8	0	8	8	0	8	8	0	8	8	2.3
5-FullIns_4	(9,–)	40	9	9	0	9	9	0	9	9	0	9	9	0	9	9	0	9	9	0	9	9	0	9	9	0	9	9	7
1-Insertions_4	(4,4)	1	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0
1-Insertions_5	(–,6)	5	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	1.2
1-Insertions_6	(–,7)	19	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0.2
2-Insertions_4	(4,4)	3	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	1
2-Insertions_5	(–,6)	15	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0.2
3-Insertions_4	(–,5)	6	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	1.2
3-Insertions_5	(–,6)	37	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	0	6	6	6.1
4-Insertions_3	(3,3)	2	4	4	0	4	4	0	4	4	0	4	4	0	4	4	0	4	4	0	4	4	0	4	4	0	4	4	0.1
4-Insertions_4	(–,7)	11	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0.3
DSJC125.1	(–,5)	10	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	6	0	6	6	0	5	6	0	5	6	0
DSJC125.5	(–,17)	14	17	17	0.5	17	17	1.3	17	17	1.6	17	17	6.6	18	18	0	18	18	1	18	18	0.4	18	19	2.1	18	18	2.5
DSJC125.9	(–,30)	12	44	44	0.1	44	44	0.1	44	44	0.1	44	44	0.2	44	44	0.3	44	44	0.3	44	44	2	44	44	9.9	44	45	3.4
DSJC250.1	(–,8)	30	8	8	0.1	8	8	0	8	8	0.2	8	8	0.3	8	9	0	9	9	0	9	9	0.2	9	9	0	9	9	24.4
DSJC250.5	(–,22)	47	28	28	22.3	28	28	30.8	28	28	33.6	29	29	2.8	29	30	0.9	31	31	1.9	29	30	2.7	32	32	6.2	29	30	5.4
DSJC250.9	(–,72)	60	72	72	3.8	72	72	28.2	72	72	5.6	72	72	26.7	72	73	6.3	73	74	1.6	72	72	26.6	74	74	39	75	77	12
DSJC500.1	(–,12)	37	13	13	0.1	13	13	0.1	13	13	0.1	13	13	0.2	13	13	0.2	14	14	0.1	14	14	1.6	14	14	4.5	14	14	29.9
DSJC500.5	(–,48)	168	49	50	35	50	50	100.3	50	50	105.8	50	51	20.4	52	52	81.4	55	55	113	51	51	47.3	55	55	138.5	50	50	123
DSJC500.9	(–,126)	398	127	127	234.4	128	129	180.8	127	128	82.3	128	129	127	129	130	154	133	133	29.1	127	128	377.2	134	135	340.3	132	132	204.4
DSJC1000.1	(–,20)	174	21	21	2.8	21	21	164.3	21	21	5.9	21	21	10.4	21	22	0.8	23	23	5.4	23	23	46.7	23	23	90.5	22	22	169.9
DSJC1000.5	(–,83)	1102	89	90	309.7	89	90	962.5	90	91	303.5	90	91	496.9	93	93	546.3	99	99	515	90	91	409.7	97	98	981.1	86	86	514.8
DSJC1000.9	(–,224)	2693	226	227	1983.5	230	232	1869.2	227	228	2245	230	230	2382.1	233	234	1621.1	238	238	1447.8	226	229	2401.4	245	247	2143.7	232	233	125.9
DSJR500.1c	(–,63)	171	86	87	149.6	86	86	20.1	85	85	94.3	89	92	1.1	85	85	22.6	88	92	1.2	89	89	133.6	87	90	72.6	91	92	0.2
DSJR500.5	(–,26)	124	126	127	18	126	126	86.1	126	127	54.2	128	128	42.1	122	123	69.4	128	129	5.6	127	128	65.9	129	130	10.8	129	131	55
abb313GPIA	(–,–)	328	9	9	4.1	9	9	26.4	9	9	0.9	9	9	30.7	9	9	1.1	10	11	0.1	11	11	0	11	11	0.1	12	13	36.4
ash331GPIA	(–,4)	200	4	4	0	4	4	0	4	4	0	4	4	0	4	4	0	4	4	0	4	4	0	4	4	0	5	5	25.8
ash608GPIA	(–,–)	633	4	4	0.1	4	4	0.2	4	4	0.1	4	4	0.2	4	4	0.1	4	4	0.3	4	5	0	4	4	409.3	5	5	27.4
ash958GPIA	(–,–)	1627	4	4	0.5	4	4	0.5	4	4	0.6	4	4	0.9	4	4	0.4	4	4	1.3	5	5	0	4	5	0	5	5	121.5
will199GPIA	(–,7)	31	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	8	43.9
flat300_20_0	(20,20)	112	20	20	0.3	20	20	0.3	20	20	0.4	20	20	0.6	20	20	0.6	22	23	0.4	20	20	1.1	33	34	76.9	20	20	2.9
flat300_26_0	(26,26)	89	26	26	5.5	26	26	16.1	26	26	16.6	26	26	19.8	33	33	4.3	29	31	56.5	32	33	4.3	35	35	53.5	33	34	2.9
flat300_28_0	(28,31)	61	31	32	3.4	31	31	54.6	31	32	3.3	31	32	7.8	33	33	7.2	35	35	1.6	33	33	5.1	35	36	17.3	33	34	2.9
flat1000_50_0	(50,50)	1076	85	86	957.4	50	78	1004.9	88	88	729.5	87	88	713.3	50	50	636.3	54	54	280.3	86	88	470.3	95	96	939.4	84	86	359.3
flat1000_60_0	(60,60)	1119	88	89	245.2	87	88	918.5	89	90	128.2	89	90	372.3	90	91	719.2	64	65	565.1	88	89	1014.6	96	97	624.5	87	87	235.4
flat1000_76_0	(76,83)	1147	88	89	618.1	88	89	957.6	89	90	188.7	90	90	712	92	92	605	98	98	553.3	89	90	399	96	97	869	87	87	306
latin_square_10	(–,99)	1242	103	104	617.8	106	107	889.4	103	104	510.4	104	105	458.4	102	103	214.9	105	106	404.5	101	102	369.9	111	114	798.4	117	118	970.7
qg.order100	(100,–)	12102	100	100	17.9	100	100	18.5	100	100	18.3	100	100	19.9	100	100	36.6	100	100	300.6	100	101	14.8	100	100	875.1	100	101	3971.9
le450_5a	(5,5)	230	5	5	0.1	5	5	0.1	5	5	0.1	5	5	0.9	5	5	0.2	6	7	0	5	7	0	6	7	0	6	7	47.4
le450_5b	(5,5)	232	5	5	0.3	5	5	0.5	5	5	0.6	5	5	0.6	5	5	0.3	6	7	0	6	7	0	6	6	59.6	7	7	37.1
le450_5d	(5,5)	191	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	5	0	5	6	16.4
le450_15a	(15,–)	68	15	15	0.2	15	15	3.4	15	15	0.1	15	15	15	15	15	2.2	16	16	0	16	16	0	16	16	0	16	17	9.3
le450_15b	(15,15)	76	15	15	0.1	15	15	0.2	15	15	0.1	15	15	5.8	15	15	0.3	16	16	0	16	16	0	16	16	0	16	16	18.9
le450_15c	(15,15)	45	16	16	13.5	15	15	19.5	15	15	19.1	15	16	8	15	15	5.9	18	18	2.3	23	23	0	23	23	0	19	21	216.4
le450_15d	(15,15)	42	16	16	21.7	15	16	13	15	15	20.3	15	16	7.1	15	15	7.8	18	19	2.1	22	23	0	22	23	0	20	21	189.6
le450_25c	(25,26)	56	26	26	0.7	26	27	0	26	26	2	26	27	0.1	26	26	18	27	28	0	27	28	0	27	28	0	27	28	32.3
le450_25d	(25,26)	59	26	26	0.5	26	27	0	26	26	0.8	26	27	0.2	26	26	4.7	28	28	0	28	28	0	27	28	0	27	27	38.1
queen6_6	(7,7)	2	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	7	7	1.5	7	7	0
queen7_7	(7,7)	4	7	7	0	7	7	0	7	7	0	7	7	0	7	7	0	8	8	0	7	7	0	7	7	2.1	7	7	0
queen8_12	(12,–)	7	12	12	0	12	12	0	12	12	0	12	12	0	12	12	0	12	12	0	12	12	0.1	12	12	0.4	12	12	0.9
queen8_8	(9,9)	5	9	9	0	9	9	0	9	9	0	9	9	0	9	9	0	9	9	0.1	9	9	0.1	9	10	0	9	9	0.2
queen9_9	(10,10)	6	10	10	0	10	10	0	10	10	0	10	10	0	10	10	0	10	10	0.4	10	10	0.1	10	11	0	10	10	0.4
queen10_10	(11,11)	10	11	11	0.1	11	11	0.1	11	11	0	11	11	0.1	11	11	0.8	12	12	0	11	12	0.1	12	12	0.1	11	11	0.9
queen11_11	(11,12)	14	12	12	0.1	12	12	0.1	12	12	0.2	12	12	0.2	12	13	0	13	13	0	12	13	0.1	13	13	0.4	12	12	2
queen12_12	(12,12)	18	13	13	1	13	13	1.4	13	13	0.9	13	13	3.8	13	14	0	14	14	0	14	14	0.2	14	14	1.1	13	13	13.3
queen13_13	(13,14)	22	14	14	2.9	14	14	2.3	14	14	1.3	14	14	13.2	15	15	0	15	15	0.2	15	15	0.3	15	15	2.7	14	14	21.4
queen14_14	(14,–)	21	15	16	0	15	16	0	15	15	20	15	16	0	16	16	0	16	16	3.1	16	16	0.5	16	16	5	15	15	32.2
queen15_15	(15,17)	24	16	17	0	16	17	0	16	16	23.9	16	17	0	17	17	0	18	18	0	17	17	0.8	17	17	5.7	16	17	23.9
queen16_16	(16,18)	24	18	18	0	18	18	0	18	18	0	18	18	0	18	18	0	19	19	0	17	18	1.2	18	18	19.5	17	17	33.4
school1	(–,–)	142	14	14	0.1	14	14	0.1	14	14	0.1	14	14	0.2	14	14	0.2	16	17	0.2	14	14	1.6	14	14	65.3	14	17	20.3
school1_nsh	(–,14)	105	14	14	0.1	14	14	0.1	14	14	0.1	14	14	0.1	14	14	0.2	16	17	0.1	14	14	0.4	18	23	12	19	22	2
wap01a	(–,–)	412	43	44	1.2	43	44	1.6	43	44	1.5	42	42	217.1	42	42	55	44	45	0.9	44	45	107.8	44	46	30.8	47	48	131.6
wap02a	(40,–)	318	42	43	0.7	42	43	0.8	42	42	251.6	41	42	4.9	41	41	159.9	43	43	0.8	43	43	97.8	43	44	0.5	46	47	150.8
wap03a	(–,–)	1395	46	47	3.8	46	47	4.5	46	46	365.2	45	47	5.8	44	44	782	48	48	4.2	46	47	198.9	47	48	339.1	50	51	884.3
wap04a	(40,–)	2125	44	44	170	45	45	2.8	44	44	484.3	43	44	31.2	43	43	833.6	45	46	1.7	45	46	1.6	45	46	1.8	47	49	1073.5
wap06a	(40,–)	138	41	42	0.5	42	42	0.7	42	42	0.5	42	43	0.2	40	41	8.1	42	43	0.2	42	44	0.2	42	43	4.7	44	44	24.3
wap07a	(–,–)	341	43	44	0.7	43	43	1.9	43	44	0.7	42	43	30.9	42	42	215.2	45	45	0.6	44	45	9.1	44	45	39	47	47	80.7
wap08a	(40,–)	373	42	43	10.3	42	43	1.4	43	43	56.1	42	44	0.7	42	42	41.4	44	45	0.5	45	45	0.4	44	45	0.4	46	47	89.4

Table C.4.: Numerical results on the DIMACS graph colouring instances.

$\|V\|$	ρ	$\bar{r}$	t_{uv}	t_{uu}	Time (sec.)	st. dev. (sec.)	Predicted (sec.)
60	0.1	3	3	3	14.51	3.07	10
			5	5	68.28	19.64	20
		5	3	3	137.78	28.27	67
			5	5	–	–	454
	0.5	3	3	3	78.34	9.21	30
			5	5	357.77	48.91	321
		5	3	3	704.17	164.04	638
			5	5	–	–	2544
	0.9	3	3	3	146.31	18.98	117
			5	5	795.61	111.75	637
		5	3	3	1750.68	333.17	1204
			5	5	–	–	4615
120	0.1	3	3	3	72.91	15.12	35
			5	5	–	–	339
		5	3	3	–	–	669
			5	5	–	–	2658
	0.5	3	3	3	408.90	37.50	483
			5	5	–	–	1976
		5	3	3	–	–	3601
			5	5	–	–	13389
	0.9	3	3	3	816.97	73.84	926
			5	5	–	–	3599
		5	3	3	–	–	6505
			5	5	–	–	24021
240	0.1	3	3	3	449.70	59.28	507
			5	5	–	–	2066
		5	3	3	–	–	3760
			5	5	–	–	13973
	0.5	3	3	3	2717.32	199.17	2804
			5	5	–	–	10473
		5	3	3	–	–	18811
			5	5	–	–	69061
	0.9	3	3	3	5358.58	513.57	5080
			5	5	–	–	18803
		5	3	3	–	–	33724
			5	5	–	–	123643

Table C.5.: Uniform random graphs. Each row comprises 10 graphs and the values reported are average values over these graphs. On the smallest instances (the first two lines) we adjusted the value returned by the prediction model as this was less or equal 2 seconds.

C.2. Set T-colouring

C.2.1. Instance statistics and computation times

In the Tables C.5 and C.6 for the Uniifrom and Geometric random graph, repsectively, we report the edge density, and the average requirement, vertex distance, and edge distance. Moreover we give the average and standard deviation for the computation time of GOF-TS obtained by 10 observations per graph/class. These data are used for deriving the linear regression model reported in Table 5.3. The value of the predicted time computed through the selected model are also reported in the table. The predicted times, less a factor of 10, are used for the time limits in the main experiments of Section 5.6.7.

C.2.2. Validation of our re-implementation of tabu search

In the following tables about the set T-colouring problem we report k as the number of colours used in the solution. The span which is the objective to minimise can then be derived as $sp(G) = k - 1$.

In Table C.7, we compare our implementation of GOF-TS with the implementation of Dorne and Hao (1998b) on the instances they used for presenting results of their algorithm. Some of these instances are computationally demanding and they were not used in the main experiment for the comparison of SLS algorithms (see Section 5.6.7). The results in the Table were obtained by only one run of our implementation of GOF-TS with $I_{max} = 10000 \times \sum_{v \in V} r(v)$ iterations as stopping criterion. Dorne and Hao (1998b) run

Instance	$\|V\|$	ρ	$\bar{r}$	$\bar{t}_{uv}$	$\bar{t}_{uu}$	Time (sec.)	st. dev. (sec.)	Predicted (sec.)
GEOM60	60	0.14	5.25	10	3.97	59.20	0.58	71
GEOM70	70	0.14	5.49	10	4.22	138.91	2.24	106
GEOM80	80	0.14	5.81	10	4.32	162.71	6.51	149
GEOM90	90	0.13	5.89	10	4.32	158.86	13.97	181
GEOM100	100	0.13	5.81	10	4.48	174.60	1.41	217
GEOM110	110	0.12	5.85	10	4.52	376.36	15.18	251
GEOM120	120	0.13	5.67	10	4.56	214.54	52.73	273
GEOM30a	30	0.26	5.70	10	3.96	37.93	0.60	38
GEOM40a	40	0.24	5.08	10	4.27	44.23	1.26	50
GEOM50a	50	0.24	6.04	10	4.57	126.96	11.18	108
GEOM60a	60	0.23	6.03	10	4.64	139.64	26.19	142
GEOM70a	70	0.22	5.41	10	4.73	143.03	9.61	147
GEOM80a	80	0.22	4.86	10	4.80	214.72	38.09	151
GEOM90a	90	0.22	5.04	10	4.80	154.99	3.57	191
GEOM100a	100	0.22	5.28	10	4.87	221.09	25.65	250
GEOM110a	110	0.22	5.47	10	4.91	294.62	34.68	312
GEOM120a	120	0.22	5.53	10	4.97	380.98	47.29	369
GEOM20b	20	0.27	2.00	10	2.56	2.70	0.03	3
GEOM30b	30	0.26	2.30	10	3.49	5.68	0.09	8
GEOM40b	40	0.25	2.10	10	3.98	14.37	1.19	14
GEOM50b	50	0.24	2.08	10	4.17	24.47	0.61	20
GEOM60b	60	0.24	2.12	10	4.39	28.62	1.75	30
GEOM70b	70	0.23	2.11	10	4.50	36.06	0.92	38
GEOM80b	80	0.24	2.11	10	4.60	33.19	0.50	49
GEOM90b	90	0.24	2.04	10	4.72	66.01	4.40	59
GEOM100b	100	0.23	2.00	10	4.77	90.48	1.78	69
GEOM110b	110	0.23	2.00	10	4.79	79.92	9.13	79
GEOM120b	120	0.23	1.96	10	4.87	94.81	3.24	91

Table C.6.: The Geometric graphs. Each row represents one single instance and the values of computation time are obtained by averaging over 10 runs.

				Dorne and Hao		GOF-TS				
inst	$\bar{r}$	$\bar{t}_{uv}$	$\bar{t}_{vv}$	k	iter. ($\times 10^7$)	k	iter. ($\times 10^7$)	Time (min.)	Exit Time (min.)	Predicted (min.)
essai.30.86.10	2.87	2.50	1.90	27	1	27	0.09	0.0	0.1	10
essai.100.275.10	2.75	2.95	2.54	46	1	47	0.28	0.1	0.5	10
essai.300.937.10	3.12	3.09	2.82	81	2	80	0.94	4.9	13.7	19
essai.500.1507.10	3.01	3.04	2.89	113	2	107	1.51	36.0	48.6	55
essai.1000.3049.10	3.05	2.98	2.94	179	3(?)	174	3.05	275.1	469.7	294
essai.30.95.50	3.17	2.90	2.61	62	1	63	0.10	0.1	0.2	10
essai.100.304.50	3.04	2.93	2.89	115	1	116	0.30	2.9	4.2	5
essai.300.905.50	3.02	2.91	2.96	268	2	276	0.91	75.0	81.8	78
essai.500.1484.50	2.97	2.93	2.98	403	2	410	1.48	119.2	338.8	253
essai.1000.3024.50	3.02	3.04	2.99	847	3(?)	788	3.02	1669.4	1979.0	1459
essai.30.90.90	3.00	3.37	2.83	103	1	105	0.09	0.3	0.6	10
essai.100.299.90	2.99	3.00	2.91	225	1	233	0.30	1.3	8.6	10
essai.300.940.90	3.13	2.98	2.97	573	2	602	0.94	153.0	165.8	168
essai.500.1536.90	3.07	2.97	2.99	932	2	934	1.54	57.9	676.9	523
essai.1000.2975.90	2.98	2.98	3.00	1724	3(?)	1736	–	935.9	–	2395

Table C.7.: Statistics for the `essai` graphs and comparison of our implementation of GOF-TS with the implementation of Dorne and Hao (1998b) on the random graphs. Results for our implementation are taken from one single run with I_{max} as stopping criterion while the results of Dorne and Hao (1998b) are the best from 1, 2, or 3 runs. Note that computation times are expressed in minutes. In the last column, we report for reference also the computation time needed GOF-TS to perform I_{max} iterations as predicted by the model given in Table 5.3. The symbols $\bar{r}$, $\bar{t}_{uv}$, and $\bar{t}_{vv}$ indicate the average vertex colour requirement, edge-distance, and vertex-distance, respectively.

instead their algorithm for 10^7 iterations, or, for the largest instances, 2 or 3×10^7. Times are hardly comparable because the results of a benchmark code were not reported in that publication. However, given the difference in the number of maximum iterations we are confident that our implementation is not worse than the original one (the Wilcoxon test run on these results does not reject the hypothesis that the two algorithms perform the same).

C.2.3. Detailed results

In Table C.8 we report the numerical results of the experiment presented in Figure 5.18. For each set of parameters, indicated by the label of the class, 10 graphs with different random seed were used. For each of these classes we give the lower bound, the maximum time allowed in our experiments, and the minimum, median, and maximum value for k found by the respective algorithms. The variance of these values is determined by two factors: the stochastic algorithm and the graph. Within each class, we run the Friedman test to check whether algorithms perform differently. In the affirmative case, an all-pairwise Wilcoxon test with Holm's adjustment of the p-value is performed to check whether there is statistical significance in the differences. The grey background in the Figure indicates all algorithms that are not significantly different from the best one. We observe that the Wilcoxon test is not very powerful in distinguishing differences due to the low sample sizes (only 10 for each test). In contrast, the Friedman procedure presented in Figure 5.18 was able to indicate many more significant differences.

Table C.9 presents the results relative to the random graphs discussed in Figure 5.15. In this case, each row corresponds to the indicated instance. For each instance, 3 runs per algorithm were done (except for G-DSATUR for which we did 10 runs). We report the minimum, median, and maximum value for k. No alternative statistical test to the one discussed in Section 5.6.7 is reported in this case because 3 runs per instance are too few to detect any difference by means of the Wilcoxon test and the appropriate rank-based procedure in this case must necessarily aggregate results over the instances as done with the Friedman test.

Instance	Lwb(sec)	G-DSATUR			max time	GOF-AG			GOF-TS-reass			GOF-TS			GSF-MC			GSF-TS			GSF-HEA			GSV-TS			GSF-GLS			FCNS		
T-G.5.5-60.0.1	35(2)	41	48	62	90	34	36	39	36	39	46	36	39	46	36	39.5	46	36	38.5	46	36	39	46	36	39	46	43	50	55	37	41	46
T-G.5.10-60.0.1	55(2)	78	89	107	20	58	69.5	83	66	76.5	95	65	76.5	83	73	80.5	94	63	77.5	96	63	74	83	66	73	94	81	95	111	65	76	85
T-G.10.5-60.0.1	72(7)	79	89	116	710	63	70	92	71	88	98	71	78.5	94	72	84	98	74	86	98	72	77.5	92	72	88.5	102	84	93	106	73	78.5	92
T-G.5.5-60.0.5	43(7)	89	100.5	116	320	76	83.5	89	74	84.5	89	75	85	89	75	85.5	90	78	89.5	94	80	91.5	96	80	89.5	94	89	103.5	116	89	101.5	110
T-G.5.10-60.0.5	79(7)	168	204	231	1455945	140	154.5	171	137	154	169	139	154	170	153	167	199	143	157	175	150	165.5	185	145	159	178	168	197	220	168	187	213
T-G.10.5-60.0.5	81(21)	148	190	222	1627348	121	147.5	161	123	149	163	123	148.5	162	132	157	172	134	162	210	133	165.5	186	142	168.5	221	149	184	222	149	183.5	210
T-G.5.5-60.0.9	82(38)	160	180	194	1250	133	156	164	137	155	164	138	156.5	165	143	158.5	166	147	164	173	148	165.5	176	147	165	174	160	176	194	160	180.5	194
T-G.5.10-60.0.9	111(45)	304	325	370	238959	264	274.5	294	261	275.5	291	264	278	294	272	286	306	277	289.5	304	284	295	319	283	295.5	309	304	320.5	370	311	322.5	335
T-G.10.5-60.0.9	156(110)	288	323	353	649891	251	274.5	297	253	276.5	298	254	278.5	301	257	280.5	303	257	283.5	302	271	302.5	329	260	297	310	295	320	353	295	319.5	353
T-G.5.5-120.0.1	35(2)	58	66	79	380	44	46	56	45	47	57	45	47.5	57	46	49	58	46	49	57	48	50.5	57	46	49	57	62	67	76	53	55.5	61
T-G.5.5-120.0.5	60(9)	154	167	189	4728	125	131.5	142	125	130.5	141	127	133	143	127	134	142	134	140	150	139	145	157	134	141	151	159	166	176	161	164	183
T-G.5.5-120.0.9	114(34)	276	303	321	8950	248	265.5	274	247	264	276	250	267	279	249	270	281	265	282.5	292	270	286.5	299	265	285	296	280	304.5	314	283	302.5	312

Table C.8.: Summary of results on random graphs. For each row of the table 10 instances with the features syntetically reported in the names of the instances (see Section 5.4) were considered. We report the minimum, median, and maximum result. Grey background in the cell is used to emphasise the best algorithms on each instance.

Instance	Lwb(sec)	Best	G-DSATUR			max time	GOF-AG			GOF-TS-reass			GOF-TS			GSF-MC			GSF-TS			GSF-HEA			GSV-TS			GSF-GLS			FCNS		
GEOM60	230	258	258	258	265	710	**257**	257	257	258	258	258	258	258	258	258	258	258	258	258	258	258	258	258	258	258	258	258	258	258	262	263
GEOM70	260	273	283	287.5	299	1060	**257**	269	269	272	273	276	269	270	270	273	276	277	273	275	276	277	278	278	278	285	285	285	285	288	277	278	280
GEOM80	365	383	392	395	395	1457	**370**	372	381	393	393	393	387	389	392	389	390	390	389	389	393	390	391	392	394	394	396	395	395	397	394	395	396
GEOM90	313	332	335	338.5	347	1742	333	334	335	333	334	334	**331**	332	333	336	336	337	334	334	336	334	334	335	335	335	336	336	336	339	336	337	337
GEOM100	378	404	412	416	420	2061	**403**	404	404	410	413	414	410	412	413	412	414	414	410	413	413	409	413	413	411	414	416	412	415	417	414	415	425
GEOM110	348	383	400	410	415	2364	**375**	376	379	384	385	386	382	383	385	388	388	390	387	388	390	400	402	404	402	408	409	404	408	410	390	390	390
GEOM120	343	402	412	419	420	2560	**396**	398	406	409	409	410	408	413	419	408	409	414	407	411	416	408	408	412	420	420	422	420	421	422	412	414	416
GEOM30a	182	209	238	238	243	380	**196**	201	211	215	219	225	214	215	221	212	215	224	219	228	228	215	217	219	234	234	238	238	238	238	212	215	217
GEOM40a	160	213	229	229	232	500	**207**	207	213	217	219	223	214	215	216	216	217	218	217	220	222	219	219	221	223	226	227	226	229	229	217	218	220
GEOM50a	199	318	335	345	345	1081	**307**	308	317	320	323	327	319	320	324	328	328	329	334	334	335	332	332	333	338	340	340	341	345	345	323	324	325
GEOM60a	290	358	369	373	379	1394	**353**	356	365	365	366	367	362	363	367	365	365	366	367	367	371	369	371	373	368	373	378	369	373	380	368	368	369
GEOM70a	425	469	487	487	487	1439	**478**	478	478	486	487	487	482	484	486	485	487	487	485	485	487	483	487	487	487	487	487	487	487	487	480	481	482
GEOM80a	241	379	388	396	396	1475	**360**	361	363	376	377	378	374	382	382	380	381	384	377	378	380	377	380	384	383	387	393	383	392	396	380	382	382
GEOM90a	285	377	398	405	405	1830	**375**	380	380	398	398	405	398	398	402	398	398	402	398	398	403	395	395	395	398	398	405	398	398	405	382	383	384
GEOM100a	302	459	462	471	475	2355	**423**	434	438	452	453	455	447	451	451	447	449	449	449	457	461	450	454	461	459	464	470	460	470	471	459	460	462
GEOM110a	385	494	523	523	523	2906	**475**	498	498	506	509	509	505	505	508	499	501	505	506	509	512	507	507	509	518	523	523	523	523	523	501	503	504
GEOM120a	514	556	571	578	583	3415	**547**	550	551	559	560	566	555	562	564	557	559	562	559	566	566	570	571	573	578	578	583	578	578	584	559	562	563
GEOM20b	39	44	45	45	45	30	**38**	43	43	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	44	45	45	45	45	45	45
GEOM30b	38	77	78	78	78	80	**66**	76	76	77	77	77	77	77	77	77	77	77	77	77	77	77	77	77	77	77	77	78	78	78	77	77	77
GEOM40b	74	74	79	86	87	140	**74**	74	74	74	74	74	74	74	74	75	76	76	75	75	76	75	76	76	75	75	76	81	86	87	77	78	79
GEOM50b	67	87	92	94	97	200	**81**	83	84	83	85	85	83	84	84	87	88	88	87	87	87	87	87	88	88	88	88	92	95	97	88	88	89
GEOM60b	79	116	123	132	138	300	**117**	118	119	118	119	121	118	119	123	121	122	123	119	120	120	121	121	122	122	122	123	129	129	132	119	120	122
GEOM70b	94	121	133	135	137	380	120	120	121	121	121	122	**119**	120	121	125	126	127	123	124	125	124	125	126	124	124	126	134	136	136	121	122	124
GEOM80b	110	141	148	149	152	490	**137**	137	138	139	140	140	139	139	140	142	142	143	141	141	141	143	143	144	141	142	142	150	153	153	144	145	146
GEOM90b	112	157	160	161	165	590	**146**	147	149	146	147	149	148	151	154	154	157	157	152	153	155	154	154	155	155	155	156	160	164	165	153	153	154
GEOM100b	133	170	173	179	183	690	**160**	160	161	163	164	165	162	162	163	170	173	174	170	171	171	169	170	171	173	173	173	173	182	183	165	166	166
GEOM110b	182	206	221	225.5	238	790	**195**	207	209	210	210	212	210	212	214	214	216	216	211	212	214	216	216	218	215	215	216	218	221	223	209	209	213
GEOM120b	172	199	206	219.5	225	910	**195**	196	198	198	199	201	197	198	204	204	205	205	202	203	203	202	203	207	203	204	205	205	217	223	198	198	200

Table C.9.: Summary of results. Given are the chromatic number, the best known result span, and the maximum time allowed for each run. Then, for each algorithm the minimum, median, and maximum span found in the trials performed (except for G-DSATUR, only 3 trials were performed and hence the three values are actually the three results). No statistical test is performed within the instances, as the number of replicates available is too low.

References

Aardal K.I., van Hoesel C.P.M., Koster A.M.C.A., Mannino C., and Sassano A. (2001). "Models and solution techniques for the frequency assignment problem". ZIB-report 01–40, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Berlin, Germany. (Cited on pages 163, 166, 167 and 173.)

Aarts E. and Lenstra J. (eds.) (1997). *Local Search in Combinatorial Optimization*. John Wiley & Sons, Chichester, UK. (Cited on pages 4, 20 and 310.)

Achlioptas D., Gomes C.P., Kautz H.A., and Selman B. (2000). "Generating satisfiable problem instances." In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence*, pp. 256–261. AAAI Press/The MIT Press. (Cited on page 80.)

Achlioptas D. and Naor A. (2005). "The two possible values of the chromatic number of a random graph". To appear in *Annals of Mathematics*. (Cited on page 89.)

Ahuja R., Ergun O., Orlin J., and Punnen A. (2002). "A survey of very large scale neighborhood search techniques". *Discrete Applied Mathematics*, 123(1-3), pp. 75–102. (Cited on pages 32 and 35.)

Ahuja R., Orlin J., Pallottino S., Scaparra M., and Scutellá M. (2004). "A multi-exchange heuristic for the single-source capacitated facility location problem". *Management Science*, 50(6), pp. 749–761. (Cited on page 155.)

Ahuja R., Orlin J., and Sharma D. (2001a). "Multi-exchange neighborhood search algorithms for the capacitated minimum spanning tree problem". *Mathematical Programming*, 91(1), pp. 71–97. Series A. (Cited on page 155.)

Ahuja R.K., Boland N.L., and Dumitrescu I. (2001b). "Exact and heuristic algorithms for the subset disjoint minimum cost cycle problem". Working Paper. (Cited on pages 107 and 111.)

Ahuja R.K., Magnanti T.L., and Orlin J.B. (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall. (Cited on page 16.)

Al-Fawzan M.A. (2000). "Algorithms for estimating the parameters of the Weibull distribution". *InterStat, Statistics on the Internet*. http://interstat.statjournals.net/. (June 2005). (Cited on page 75.)

Alimonti P. (1996). "New local search approximation techniques for maximum generalized satisfiability problems". *Information Processing Letters*, 57(3), pp. 151–158. (Cited on page 34.)

Allen M., Kumaran G., and Liu T. (2002). "A combined algorithm for graph-coloring in register allocation". In Johnson et al. (2002b), pp. 100–111. (Cited on page 85.)

Allen S.M., Smith D.H., and Hurley S. (1999). "Lower bounding techniques for frequency assignment". *Discrete Mathematics*, 197/198, pp. 41–52. (Cited on page 167.)

Anderson M. and ter Braak C. (2003). "Permutation tests for multi-factorial analysis of variance". *Journal of Statistical Computation and Simulation*, 73(2), pp. 85–113. (Cited on pages 49, 59 and 275.)

Appel K., Haken W., and Koch J. (1977). "Every planar map is four colorable". *Illinois Journal of Mathematics*, 21, pp. 429–567. (Cited on page 85.)

Applegate D., Bixby R., Chvátal V., and Cook W. (1998). "On the solution of traveling salesman problems". *Documenta Mathematica. Extra Volume. Proceedings of the International Congress of Mathematicians*, III, pp. 645–656. (Cited on page 175.)

Arnold S.F. (1981). *Theory of Linear Models and Multivariate Analysis*. John Wiley & Sons, New York, NY, USA. (Cited on page 73.)

Arntzen H. and Løkketangen A. (2003). "A tabu search heuristic for a university timetabling problem". In *Proceedings of the Fifth Metaheuristics International Conference*. Kyoto, Japan. (Cited on page 235.)

Ausiello G., Crescenzi P., Gambosi G., Kann V., Marchetti-Spaccamela A., and Protasi M. (1999). *Complexity and approximation: Combinatorial optimization problems and their approximability properties*. Springer-Verlag. (Cited on pages 14 and 215.)

Avanthay C., Hertz A., and Zufferey N. (2003). "A variable neighborhood search for graph coloring". *European Journal of Operational Research*. (Cited on page 105.)

Balakrishnan N., Lucena A., and Wong R.T. (1992). "Scheduling examinations to reduce second-order conflicts". *Computers & Operations Research*, 19(5), pp. 353–361. (Cited on page 208.)

Barbosa V.C. and Ferreira R.G. (2004). "On the phase transitions of graph coloring and independent sets". *Physica A: Statistical Mechanics and its Applications*, 343(1–2). (Cited on page 149.)

Barnes W., Dimova B., and Dokov S.P. (2003). "The theory of elementary landscapes". *Applied Mathematics Letters*. (Cited on page 78.)

Barnier N. and Brisset P. (2002). "Graph coloring for air traffic flow management". In *CPAIOR'02: Fourth International Workshop on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*, pp. 133–147. Le Croisic, France. (Cited on page 85.)

Barr R., Golden B., Kelly J., Resende M., and Stewart W. (1995). "Designing and reporting on computational experiments with heuristic methods." *Journal of Heuristics*, 1(1), pp. 9–32. (Cited on pages 4 and 39.)

Battiti R. and Protasi M. (2001). "Reactive local search for the maximum clique problem". *Algorithmica*, 29(4), pp. 610–637. (Cited on pages 101 and 287.)

Battiti R. and Tecchiolli G. (1994). "The reactive tabu search". *ORSA Journal on Computing*, 6, pp. 126–140. (Cited on page 26.)

Bellare M., Goldreich O., and Sudan M. (1998). "Free bits, PCPs, and nonapproximability—towards tight results". *SIAM Journal on Computing*, 27(3), pp. 804–915. (Cited on page 89.)

Berger B. and Rompel J. (1990). "A better performance guarantee for approximate graph colouring". *Algorithmica*, 5(4), pp. 459–466. (Cited on page 89.)

Bertsekas D.P., Tsitsiklis J.N., and Wu C. (1997). "Rollout algorithms for combinatorial optimization". *Journal of Heuristics*, 3(3), pp. 245–262. (Cited on page 21.)

Bianchi L., Birattari M., Chiarandini M., Manfrin M., Mastrolilli M., Paquete L., Rossi-Doria O., and Schiavinotto T. (2004). "Metaheuristics for the vehicle routing problem with stochastic demand". In *Parallel Problem Solving from Nature - PPSN VIII, 8th International Conference*, edited by Yao X. et al., vol. 3242 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Germany. (Cited on page 35.)

Bibel W., Andler D., da Costa O., Küppers G., and Pearson I. (2004). "Converging technologies and the natural, social and cultural world". Tech. rep., European Commission of the EU. http://europa.eu.int/comm/research/conferences/2004/ntw/pdf/sig4_en.pdf . (Cited on page 251.)

Birattari M. (2003). "The race package for R. racing methods for the selection of the best." Tech. Rep. TR/IRIDIA/2003-37, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium. (Cited on page 68.)

Birattari M. (2004a). "On the estimation of the expected performance of a metaheuristic on a class of instances. how many instances, how many runs?" Tech. Rep. TR/IRIDIA/2004-01, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium. (Cited on pages 46, 68 and 141.)

Birattari M. (2004b). *The Problem of Tuning Metaheuristics, as seen from a machine learning perspective*. Ph.D. thesis, Université Libre de Bruxelles, Brussels, Belgium. (Cited on pages 46, 68 and 256.)

Birattari M., Stützle T., Paquete L., and Varrentrapp K. (2002). "A racing algorithm for configuring metaheuristics". In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, edited by W.B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. Potter, A. Schultz, J. Miller, E. Burke, and N. Jonoska, pp. 11–18. Morgan Kaufmann Publishers, New York. (Cited on pages 5, 7, 67, 206, 216, 220 and 249.)

Blöchliger I. and Zufferey N. (2003). "A reactive tabu search using partial solutions for the graph coloring problem". In *Coloring graphs from lists with bounded size of their union: result from Dagstuhl Seminar 03391*, edited by D. Kral and J. Sgall, vol. 156 of *ITI-Series*. Department of Applied Mathematics and Institute for Theoretical Computer Science, Charles University, Prague, Czech Republic. (Cited on page 106.)

Bollobás B. (2004). *Random graphs*. Cambridge studies in advanced mathematics. Cambridge University Press, Cambridge, UK, second ed. (Cited on page 89.)

Bollobás B. and Thomason A. (1985). "Random graphs of small order". *Annals of Discrete Mathematics*, 28, pp. 251–256. (Cited on page 89.)

Borgne L. (1994). *Automatic Frequency Assignment for Cellular Networks using local search heuristics*. Master's thesis, Uppsala University. (Cited on page 185.)

Borndörfer R., Eisenblätter A., Grötschel M., and Martin A. (1998). "Frequency assignment in cellular phone networks". *Annals of Operations Research*, 76, pp. 73–93. (Cited on page 178.)

Brandstädt A., Le V.B., and Spinrad J.P. (1999). *Graph Classes: A Survey*, vol. 3 of *Monographs on Discrete Mathematics and Applications*. SIAM Society for Industrial and Applied Mathematics, Philadelphia. (Cited on page 88.)

Brélaz D. (1979). "New methods to color the vertices of a graph". *Communications of the ACM*, 22(4), pp. 251–256. (Cited on pages 94 and 97.)

Burke E.K., Beyrouthy C., Silva J.D.L., McCollum B., and McMullan P. (2004a). "SpaceMAP - applying meta-heuristics to real world space allocation problems in academic institutions". In Burke and Trick (2004), pp. 441–444. (Cited on page 208.)

Burke E.K., Bykov Y., and Petrovic S. (2001). "A multicriteria approach to examination timetabling". In Burke and Erben (2001), pp. 118–131. (Cited on page 209.)

Burke E.K. and Carter M.W. (eds.) (1998). *Practice and Theory of Automated Timetabling II, Second International Conference, PATAT 1997*, vol. 1408 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Germany. (Cited on pages 209, 301 and 320.)

Burke E.K. and De Causmaecker P. (eds.) (2003). *Practice and Theory of Automated Timetabling IV, 4th International Conference, PATAT 2002*, vol. 2740 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Germany. (Cited on pages 209, 303, 313 and 316.)

Burke E.K., Eckersley A.J., McCollum B., Petrovic S., and Qu R. (2004b). "Analysing similarity in examination timetabling". In Burke and Trick (2004), pp. 557–559. (Cited on pages 208 and 250.)

Burke E.K., Elliman D.G., and Weare R.F. (1995). "Specialised recombinative operators for timetabling problems." In *AISB Workshop on Evolutionary Computing.*, vol. 993 of *Lecture Notes in Computer Science*, pp. 75–85. Springer Verlag, Berlin, Germany. (Cited on page 219.)

Burke E.K. and Erben W. (eds.) (2001). *Practice and Theory of Automated Timetabling III, Third International Conference, PATAT 2000*, vol. 2079 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Germany. (Cited on pages 209, 300, 301 and 320.)

Burke E.K., Kendall G., and Soubeiga E. (2003). "A tabu-search hyperheuristic for timetabling and rostering". *Journal of Heuristics*, 9(6), pp. 451–470. (Cited on pages 208 and 250.)

Burke E.K. and Ross P. (eds.) (1996). *Practice and Theory of Automated Timetabling, First International Conference, PATAT 1995*, vol. 1153 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Germany. (Cited on page 209.)

Burke E.K. and Trick M. (eds.) (2004). *Proceedings of The 5th International Conference on the Practice and Automated Timetabling,PATAT 2004*. Pittsburgh, PA. (Cited on pages 209, 300, 301, 302, 311 and 315.)

Cahit I. (2004). "Spiral chains: A new proof of the four color theorem". Unpublished. (Cited on page 88.)

Capone A. and Trubian M. (1999). "Channel assignment problem in cellular systems: a new model and a tabu search algorithm". *IEEE Transactions on Vehicular Technology*, 48(4), pp. 1252–1260. (Cited on page 167.)

Caramia M. and Dell'Olmo P. (2002a). "k-fullins graphs: a proposal of a new class of benchmarks". Manuscript. (Cited on page 92.)

Caramia M. and Dell'Olmo P. (2002b). "k-insertions graphs: a proposal of a new class of benchmarks". Manuscript. (Cited on page 92.)

Caramia M. and Dell'Olmo P. (2002c). "Vertex coloring by multistage branch and bound". In Johnson et al. (2002b), pp. 40–47. (Cited on pages 95 and 129.)

Carrasco M.P. and Pato M.V. (2001). "A multiobjective genetic algorithm for the class/teacher timetabling problem". In Burke and Erben (2001), pp. 3–7. (Cited on page 209.)

Carter M.W. and Laporte G. (1998). "Recent developments in practical course timetabling". In Burke and Carter (1998), pp. 3–19. (Cited on page 205.)

Carter M.W., Laporte G., and Lee. S.Y. (1996). "Examination timetabling: Algorithmic strategies and applications." *Journal of the Operational Research Society*, 47, pp. 373–383. (Cited on page 219.)

Carter M.W. and Price C.C. (2000). *Operations Research: a Practical Introduction*. CRC Press LLC, New Yourk, USA. (Cited on page 15.)

Castelino D., Hurley S., and Stephens N. (1996). "A tabu search algorithm for frequency assignment". *Annals of Operations Research*, 63, pp. 301–320. (Cited on pages 167 and 188.)

Chand A. (2004). "A constraint based generic model for representing complete university timetabling data". In Burke and Trick (2004), pp. 125–150. (Cited on page 207.)

Cheeseman P., Kanefsky B., and Taylor W.M. (1991). "Where the really hard problems are". In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, edited by J. Mylopoulos and R. Reiter, pp. 331–337. Morgan Kaufmann Publishers, San Francisco, CA, USA. (Cited on pages 93 and 149.)

Chiarandini M., Dumitrescu I., and Stützle T. (2003). "Local search for the graph colouring problem. A computational study." Tech. Rep. AIDA–03–01, FG Intellektik, FB Informatik, Technische Universität Darmstadt, Darmstadt, Germany. (Cited on page 119.)

Chiarandini M. and Stützle T. (2002). "An application of iterated local search to graph coloring". In Johnson et al. (2002b), pp. 112–125. (Cited on pages 119 and 122.)

Chiarandini M. and Stützle T. (2002). "Experimental evaluation of course timetabling algorithms". Tech. Rep. AIDA–02–05, FG Intellektik, FB Informatik, Technische Universität Darmstadt, Darmstadt, Germany. (Cited on pages 75 and 219.)

Coello C.A.C., Aguirre A.H., and Zitzler E. (eds.) (2005). *Evolutionary Multi-Criterion Optimization: Third International Conference, EMO 2005*, vol. 3410 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Germany. (Cited on page 70.)

Cohen P.R. (1995). *Empirical Methods for Artificial Intelligence*. MIT Press, Boston. (Cited on pages 4, 39, 49 and 244.)

Conover W. (1999). *Practical Nonparametric Statistics*. John Wiley & Sons, New York, NY, USA, third ed. (Cited on pages 51, 54, 55, 57, 61, 62, 75 and 235.)

Cornuéjols G., Liu X., and Vušković K. (2003). "A polynomial algorithm for recognizing perfect graphs". In *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, pp. 20–27. Computer Society Press. (Cited on page 90.)

Costa D. (1993). "On the use of some known methods fo T-colorings of graphs". *Annals of Operations Research*, 41, pp. 343–358. (Cited on pages 167, 172, 178 and 188.)

Coudert O. (1997). "Exact coloring of real-life graphs is easy". In *DAC '97: Proceedings of the 34th annual conference on Design automation*, pp. 121–126. ACM Press, New York, NY, USA, New York, NY, USA. (Cited on pages 94 and 129.)

Cozzens M.B. and Roberts F.S. (1982). "t-colorings of graphs and the channel assignment problem". *Congressus Numerantium*, 35, pp. 191–208. (Cited on pages 165 and 176.)

Crainic T.G., Toulouse M., and Gendreau M. (1997). "Toward a taxonomy of parallel tabu search heuristics". *INFORMS Journal on Computing*, 9(1), pp. 61–72. (Cited on page 35.)

Crescenzi P. and Viggo K. (2004). "A compendium of NP optimization problems". Visited February 2005. (Cited on pages 14 and 89.)

Culberson J. (1992). "Iterated greedy graph coloring and the difficulty landscape". Tech. Rep. 92-07, Department of Computing Science, The University of Alberta, Edmonton, Alberta, Canada. (Cited on pages 29, 149, 190 and 220.)

Culberson J. (2001). "Hidden solutions, tell-tales, heuristics and anti-heuristics." In *The IJCAI-01 Workshop on Empirical Methods in Artificial Intelligence*, edited by H. Hoos and T. Stuëtzle, pp. 9–14. (Cited on page 98.)

Culberson J., Beacham A., and Papp D. (1995). "Hiding our colors". In *Proceedings of the CP'95 Workshop on Studying and Solving Really Hard Problems*, pp. 31–42. Cassis, France. (Cited on pages 90, 91, 95, 100, 143 and 168.)

Culberson J. and Gent I.P. (1999). "Well out of reach: Why hard problems are hard." Tech. Rep. APES–13–1999, APES (Algorithms, Problems, and Empirical Studies) Group. (Cited on page 149.)

Culberson J. and Gent I.P. (2001). "Frozen development in graph coloring". *Theoretical Computer Science*, 265(1-2). (Cited on page 149.)

Culberson J. and Luo F. (1996). "Exploring the k-colorable landscape with iterated greedy". In Johnson and Trick (1996), pp. 245–284. (Cited on page 138.)

Custers N., Causmaecker P.D., Demeester P., and den Berghe G.V. (2004). "Semantic components for timetabling". In Burke and Trick (2004), pp. 169–182. (Cited on page 207.)

da Fonseca V.G., Fonseca C.M., and Hall A.O. (2001). "Inferential performance assessment of stochastic optimisers and the attainment function". In *Proceedings of Evolutionary Multi-Criterion Optimization First International Conference, EMO 2001*, edited by

C. Coello, D. Corne, K. Deb, L. Thiele, and E. Zitzler, vol. 1993 of *Lecture Notes in Computer Science*, pp. 213–224. Springer Verlag, Berlin, Germany. (Cited on pages 65, 69 and 70.)

Daniel W.W. (1978). *Applied Nonparametric Statistic*. Houghton Mifflin Company, Boston. (Cited on page 57.)

Dannenbring D.G. (1977). "Procedures for estimating optimal solution values for large combinatorial problems". *Management Science*, 23(12), pp. 1273–1283. (Cited on pages 41 and 277.)

Daskalaki S., Birbas T., and Housos E. (2004). "An integer programming formulation for a case study in university timetabling". *European Journal of Operational Research*, 153(1), pp. 117–135. (Cited on page 208.)

Davis L. (ed.) (1991). *The Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, NY. (Cited on page 30.)

de Werra D. (1985). "An introduction to timetabling". *European Journal of Operational Research*, 19(2), pp. 151–162. (Cited on pages 85, 208 and 212.)

de Werra D. (1990). "Heuristics for graph coloring". *Computing Supplement*, 7, pp. 191–208. (Cited on pages 98 and 164.)

Dean A. and Voss D. (1999). *Design and Analysis of experiments*. Springer Texts in Statistics. Springer-Verlag, New York, USA. (Cited on pages 44, 45, 50, 51, 53, 56, 58, 63 and 68.)

Demetrescu C., Finocchi I., and Italiano G.F. (2003). "Algorithm engineering, algorithmics column." *Bulletin of the EATCS*, 79, pp. 48–63. (Cited on page 217.)

den Besten M.L. (2004). *Simple Metaheuristics for Scheduling: An empirical investigation into the application of iterated local search to deterministic scheduling problems with tardiness penalties*. Ph.D. thesis, Darmstadt University of Technology, Darmstadt, Germany. (Cited on page 221.)

Di Gaspero L. (2002). *Local Search Techniques for Scheduling Problems: Algorithms and Software Tools.* Ph.D. thesis, Computer Science, Università degli Studi di Udine. (Cited on pages 207 and 218.)

Di Gaspero L. and Schaerf A. (2003a). "EasyLocal++: an object-oriented framework for the flexible design of local-search algorithms". *Software – Practice & Experience*, 33(8), pp. 733–765. (Cited on pages 3 and 127.)

Di Gaspero L. and Schaerf A. (2003b). "Multi-neighborhood local search with application to course timetabling". In Burke and De Causmaecker (2003). (Cited on page 23.)

Diaz I.M. and Zabala P. (2002). "A branch-and-cut algorithm for graph coloring". In Johnson et al. (2002b), pp. 55–62. (Cited on page 95.)

Diestel R. (2000). *Graph Theory*. Springer-Verlag, New York, Berlin, second ed., electronic ed. (Cited on pages 88 and 115.)

Dimitriou T. and Impagliazzo R. (1996). "Towards an analysis of local optimization algorithms". In *STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 304–313. ACM Press, New York, NY, USA. (Cited on page 77.)

Dorigo M. and Stützle T. (2004). *Ant Colony Optimization*. MIT Press, Cambridge, MA, USA. (Cited on page 30.)

Dorne R. and Hao J. (1998a). "A new genetic local search algorithm for graph coloring". In *Parallel Problem Solving from Nature - PPSN V, 5th International Conference*, edited by A.E. Eiben, T. Bäck, M. Schoenauer, and H.P. Schwefel, vol. 1498 of *Lecture Notes in Computer Science*, pp. 745–754. Springer Verlag, Berlin, Germany. (Cited on pages 118, 119, 126 and 253.)

Dorne R. and Hao J. (1998b). "Tabu search for graph coloring, T-colorings and set T-colorings." In *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization.*, edited by S. Voss, S. Martello, I. Osman, and C. Roucairol, chap. 6. Kluwer Academic Publishers. (Cited on pages 162, 167, 168, 175, 189, 191, 199, 255, 293 and 294.)

Duin C. and VoßS. (1999). "The pilot method: A strategy for heuristic repetition with application to the steiner problem in graphs". *Networks*, 34(3), pp. 181 – 191. (Cited on page 21.)

Dumitrescu I. (2002). *Constrained path and cycle problems*. Ph.D. thesis, The University of Melbourne. (Cited on pages 111, 155 and 253.)

Dumitrescu I. and Stützle T. (2003). "Combinations of local search and exact algorithms". In *Applications of Evolutionary Computing*, edited by G.R. Raidl et al., vol. 2611 of *Lecture Notes in Computer Science*, pp. 211–223. Springer Verlag, Berlin, Germany. (Cited on page 33.)

Eisenblätter A., Grötschel M., and Koster A.M.C.A. (2002). "Frequency assignment and ramifications of coloring". *Discussiones Mathematicae Graph Theory*, 22, pp. 51–88. (Cited on pages 162, 163, 167 and 173.)

Ekenstierna M. (2004). "Multiple comparison procedures based on marginal p-values". Tech. Rep. 2004:12, Matematiska Institutionen, Uppsala Universitet. (Cited on page 50.)

Erdős P. (1961). "Graph theory an probability II". *Canadian Journal of Mathematics*, 13, pp. 346–352. (Cited on pages 89 and 115.)

Ernst A.T., Jiang H., Krishnamoorthy M., and Sier D. (2004). "Staff scheduling and rostering: A review of applicationss, methods and models". *European Journal of operational Research*, 153(1), pp. 3–27. (Cited on page 207.)

Faigle U. and Kern W. (1991). "Note on the convergence of simulated annealing algorithms". *SIAM Journal on Control and Optimization*, 29(1), pp. 153–159. (Cited on page 33.)

Faigle U. and Kern W. (1992). "Some convergence results for probabilistic tabu search". *ORSA Journal on Computing*, 4(1), pp. 32–37. (Cited on page 33.)

Feige U. and Kilian J. (1998). "Zero knowledge and the chromatic number". *Journal of Computer and System Sciences*, 57(2), pp. 187–199. (Cited on page 89.)

Fielding M. (2000). "Simulated annealing with an optimal fixed temperature". *SIAM Journal on Optimisation*, 11(2), pp. 289–307. (Cited on page 26.)

Fink A. and Voß S. (2002). "A heuristic optimization framework". In *Optimization Software Class Libraries*, edited by S.V. and D.L. Woodruff, pp. 81–154. Kluwer Academic Publishers, Boston, MA, USA. (Cited on page 218.)

Finos L., Pesarin F., and Salmaso L. (2003). "Confronti multipli tramite metodi di permutazione". *Statistica Applicata (Italian Journal of Applied Statistics)*, 15(2), pp. 275–300. (Cited on page 275.)

Fischetti M. and Lodi A. (2003). "Local branching". *Mathematical Programming*, 98, pp. 23–47. (Cited on page 33.)

Fleischer R., Moret B., and Schmidt E.M. (eds.) (2002). *Experimental Algorithmics: From Algorithm Design to Robust and Efficient Software*, vol. 2547 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Germany. (Cited on pages 4, 39 and 313.)

Fleurent C. and Ferland J. (1996). "Genetic and hybrid algorithms for graph coloring". *Annals of Operations Research*, 63, pp. 437–464. (Cited on pages 106, 118 and 126.)

Fonseca C.M. and Fleming P.J. (1996). "On the performance assessment and comparison of stochastic multiobjective optimizers". In *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, Lecture Notes In Computer Science, pp. 584–593. Springer Verlag, Berlin, Germany. (Cited on pages 5, 70 and 72.)

Frank J., Cheeseman P., and Stutz J. (1997). "When gravity fails: Local search topology". *Journal of Artificial Intelligence Research*, 7, pp. 249–281. (Cited on page 76.)

Freimer R. (2000). "Generalized map coloring for use in geographical information systems". In *ACM-GIS 2000, Proceedings of the Eighth ACM Symposium on Advances in Geographic Information Systems*, edited by K.J. Li, K. Makki, N. Pissinou, and S. Ravada, pp. 167–173. ACM press, New York, NY, USA. (Cited on page 88.)

Frick A., Ludwig A., and Mehldau H. (1994). "A fast adaptive layout algorithm for undirected graphs". In *Proceedings of Graph Drawing, DIMACS International Workshop, GD '94*, edited by R. Tamassia and I.G. Tollis, vol. 894 of *Lecture Notes in Computer Science*, pp. 388–403. Springer Verlag, Berlin, Germany, Berlin, Germany. (Cited on page 127.)

Friedman M. (1937). "The use of ranks to avoid the assumption of normality implicit in the analysis of variance". *Journal of the American Statistical Association*, 32, pp. 675–701. (Cited on page 57.)

Galinier P. and Hao J. (1999). "Hybrid evolutionary algorithms for graph coloring". *Journal of Combinatorial Optimization*, 3(4), pp. 379–397. (Cited on pages 118, 122, 123, 137 and 253.)

Galinier P., Hertz A., and Zuffrey N. (2002). "Adaptive memory algorithms for graph colouring". In Johnson et al. (2002b), pp. 75–82. (Cited on page 123.)

Garey M.R. and Johnson D.S. (1979). *Computers and Intractability: A Guide to the Theory of $\mathcal{NP}$-Completeness*. Freeman, San Francisco, CA, USA. (Cited on pages 14, 88, 89, 213 and 215.)

Garey M.R., Johnson D.S., and So H.C. (1976). "An application of graph coloring to printed circuit testing". *IEEE Transactions on Circuits and Systems*, 23, pp. 591–599. (Cited on page 85.)

Giaro K., Janczewski R., and Malafiejski M. (2003a). "The complexity of the t-coloring problem for graphs with small degree". *Discrete Applied Mathematics*, 129(2-3), pp. 361–369. (Cited on page 167.)

Giaro K., Janczewski R., and Malafiejski M. (2003b). "A polynomial algorithm for finding t-span of generalized cacti". *Discrete Applied Mathematics*, 129(2-3), pp. 371–382. (Cited on page 167.)

Glass C.A. and Prügel-Bennett A. (2005). "A polynomially searchable exponential neighbourhood for graph colouring". *Journal of the Operational Research Society*, 56(3), pp. 324–330. (Cited on pages 123, 155 and 258.)

Glover F. (1986). "Future paths for integer programming and links to artificial intelligence". *Computers and Operations Research*, 13(5), pp. 533–549. (Cited on page 23.)

Glover F. (1989). "Tabu search – part I". *ORSA Journal on Computing*, 1(3), pp. 190–206. (Cited on page 26.)

Glover F. (1990). "Tabu search – part II". *ORSA Journal on Computing*, 2(1), pp. 4–32. (Cited on page 26.)

Glover F. (1996). "Ejection chains, reference structures and alternating path methods for traveling salesman problems". *Discrete Applied Mathematics*, 65(1), pp. 223–253. (Cited on page 23.)

Glover F. and Hanafi S. (2002). "Tabu search and finite convergence". *Discrete Applied Mathematics*, 119(1–2), pp. 3–36. (Cited on page 33.)

Glover F. and Kochenberger G. (eds.) (2002a). *Handbook of Metaheuristics*. Kluwer Academic Publishers, Norwell, MA, USA. (Cited on pages 4 and 308.)

Glover F. and Kochenberger G. (eds.) (2002b). *Handbook of Metaheuristics*, vol. 57 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers, Norwell, MA, USA. (Cited on pages 20 and 312.)

Glover F. and Laguna M. (1997). *Tabu Search*. Kluwer Academic Publishers. (Cited on page 26.)

Glover F. and Punnen A. (1997). "The travelling salesman problem: New solvable cases and linkages with the development of approximation algorithms". *Journal of the Operational Research Society*, 48, pp. 502–510. (Cited on page 261.)

Goldberg D.E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA, USA. (Cited on pages 30 and 77.)

Golden B.L. and Alt F.B. (1979). "Interval estimation of a global optimum for large combinatorial problems". *Naval Research Logistics Quarterly*, 26, pp. 69–77. (Cited on pages 41 and 277.)

Goldwasser M.H., Johnson D.S., and McGeoch C.C. (eds.) (2002). *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, vol. 59 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, Providence, RI, USA. (Cited on pages 310 and 314.)

Gomes C. and Shmoys D. (2002). "Completing quasigroups or latin squares: A structured graph coloring problem". In Johnson et al. (2002b), pp. 22–39. (Cited on pages 85, 95, 169, 170 and 171.)

Gomes C. and Shmoys D. (2004). "Approximations and randomization to boost CSP techniques". *Annals of Operations Research*, 130(1-4), pp. 117–141. (Cited on page 33.)

Gonzalez L. and Manly B.F.J. (1998). "Analysis of variance by randomization with small data sets". *Environmetrics*, 9(1), pp. 53–65. (Cited on page 275.)

González-Velarde J. and Laguna M. (2002). "Tabu search with simple ejection chains for coloring graphs". *Annals of Operations Research*, 117(1-4), pp. 165–174. (Cited on page 105.)

Good P. (2000). *Permutations tests: a practical guide to resempling methods for testing hypoteses*. Springer Verlag, New York, NY, USA, second ed. (Cited on pages 48, 49, 59 and 73.)

Graham R.L., Knuth D.E., and Patashnik O. (1994). *Concrete Mathematics*. Addison-Wesley, Reading, MA, USA, second ed. (Cited on page 109.)

Grötschel M., Lovász L., and Schrijver A. (1981). "The ellipsoid method and its consequences in combinatorial optimization". *Combinatorica*, 1(2), pp. 169–197. (Cited on page 90.)

Grover L.K. (1992). "Local search and the local structure of NP-complete problems". *Operations Research Letters*, 12(4), pp. 235–243. (Cited on pages 77 and 78.)

Gusfield D. (2002). "Partition-distance: A problem and class of perfect graphs arising in clustering." *Information Processing Letters*, 82(3), pp. 159–164. (Cited on pages 123 and 240.)

Gutin G., Koller A., and Yeo A. (2005). "Introduction to domination analysis". Manuscript. (Cited on pages 261 and 266.)

Gutin G. and Punnen A. (eds.) (2002). *The Traveling Salesman Problem and Its Variations*. Kluwer Academic Publishers, Boston, MA, USA. (Cited on pages 307 and 310.)

Gutin G., Vainshtein A., and Yeo A. (2003). "Domination analysis of combinatorial optimization problems". *Discrete Applied Mathematics*, 129(2-3), pp. 513–520. (Cited on pages 34, 261, 266 and 274.)

Gutin G., Yeo A., and Zverovitch A. (2002). "Exponential neighborhoods and domination analysis for the TSP". In Gutin and Punnen (2002), chap. 6. (Cited on pages 34, 266 and 274.)

Gutjahr W.J. (2002). "ACO algorithms with guaranteed convergence to the optimal solution". *Information Processing Letters*, 82(3), pp. 145–153. (Cited on page 33.)

Hajek B. (1988). "Cooling schedules for optimal annealing". *Mathematics of Operations Research*, 13(2), pp. 311–329. (Cited on page 33.)

Hale W.K. (1980). "Frequency assignment: Theory and applications". *Proceedings of the IEEE*, 68(12), pp. 1497–1514. (Cited on pages 162 and 165.)

Halldórsson M.M. (1993). "A still better performance guarantee for approximate graph coloring". *Information Processing Letters*, 45(1), pp. 19–23. (Cited on page 89.)

Hansen P., Hertz A., and Kuplinsky J. (1993). "Bounded vertex colorings of graphs". *Discrete Mathmatics*, 111(1-3), pp. 305–312. (Cited on page 213.)

Hansen P. and Jaumard B. (1990). "Algorithms for the maximum satisfiability problem". *Computing*, 44, pp. 279–303. (Cited on page 26.)

Hansen P. and Mladenovic N. (2002). "Variable neighborhood search". In Glover and Kochenberger (2002a), pp. 145–184. (Cited on page 23.)

Hao J.K., Dorne R., and Galinier P. (1998). "Tabu search for frequency assignment in mobile radio networks". *Journal of Heuristics*, 4(1), pp. 47–62. (Cited on pages 167, 189 and 255.)

Hao J.K. and Perrier L. (1999). "Tabu search for the frequency assignment problem in cellular radio networks". Tech. Rep. LGI2P, EMA-EERIE, Parc Scientifique Georges Besse, Nimes, France. (Cited on page 189.)

Heitjan D.F., Manni A., and Santen R.J. (1993). "Statistical analysis of *in Vivo* tumor growth experiments". *Cancer Research*, 53(24), pp. 6042–6050. (Cited on page 72.)

Herrmann F. and Hertz A. (2002). "Finding the chromatic number by means of critical graphs". *Journal of Experimental Algorithmics*, 7, p. 10. (Cited on page 96.)

Hertz A. and de Werra D. (1987). "Using tabu search techniques for graph coloring". *Computing*, 39(4), pp. 345–351. (Cited on pages 118, 119, 126 and 253.)

Hertz A., Jaumard B., and de Aragão M.P. (1994). "Local optima topology for the k-coloring problem". *Discrete Applied Mathematics*, 49(1-3), pp. 257–280. (Cited on page 139.)

Hochbaum D. (ed.) (1996). *Approximation Algorithms for $\mathcal{NP}$-hard problems.* PWS Publishing, Boston, MA, USA. (Cited on page 14.)

Hochberg Y. and Tamhane A.C. (1987). *Multiple comparison procedures*. Wiley, New York, USA. (Cited on page 50.)

Hogg T. (1996). "Refining the phase transition in combinatorial search". *Artificial Intelligence*, 81(1–2), pp. 127–154. (Cited on page 149.)

Holland J.H. (1975). *Adaption in natural and artificial systems*. The University of Michigan Press, Ann Arbor, MI, USA. (Cited on page 30.)

Hollander M. and Wolfe D.A. (1999). *Nonparametric Statistical Methods*. John Wiley & Sons, New York, NY, USA, second ed. (Cited on pages 49, 51, 55 and 57.)

Holm S. (1979). "A simple sequentially rejectvie multiple test procedure". *Scandinavian Journal of Statistics*, 6, pp. 65–70. (Cited on page 50.)

Hooker J.N. (1996). "Needed: An empirical science of algorithms". *Operations Research*, 42(2), pp. 201–212. (Cited on pages 4 and 38.)

Hoos H. and Stützle T. (1999). "Characterising the behaviour of stochastic local search". *Artificial Intelligence*, 112(1–2), pp. 213–232. (Cited on page 238.)

Hoos H. and Stützle T. (2004). *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann Publishers, San Francisco, CA, USA. (Cited on pages 3, 4, 19, 20, 69, 71, 74 and 216.)

Hoos H.H. (1999a). "On the run-time behaviour of stochastic local search algorithms for SAT". In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pp. 661–666. AAAI Press / The MIT Press, Menlo Park, CA, USA. (Cited on page 120.)

Hoos H.H. (1999b). "SAT-encodings, search space structure, and local search performance". In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, edited by T. Dean, pp. 296–302. Morgan Kaufmann Publishers, San Francisco, CA, USA. (Cited on page 96.)

Hoos H.H. and Stützle T. (1998). "Evaluating Las Vegas algorithms — pitfalls and remedies". In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence (UAI-98)*, edited by G.F. Cooper and S. Moral, pp. 238–245. Morgan Kaufmann Publishers, San Francisco, CA, USA. (Cited on page 75.)

Hoos H.H. and Stützle T. (2000). "Local search algorithms for SAT: An empirical evaluation". *Journal of Automated Reasoning*, 24(4), pp. 421–481. (Cited on pages 75 and 120.)

Hordijk W. and Manderick B. (1995). "The usefulness of recombination." In *Proceedings of Advances in Artificial Life, Third European Conference on Artificial Life*, edited by F. Morán, A. Moreno, J.J.M. Guervós, and P. Chacón, vol. 929 of *Lecture Notes in Computer Science*, pp. 908–919. Springer Verlag, Berlin, Germany. (Cited on page 78.)

Hossain S. and Steihaug T. (2002). "Graph coloring in the estimation of mathematical derivatives". In Johnson et al. (2002b), pp. 9–16. (Cited on pages 85 and 92.)

Hougardy S. (1998). "Inclusions between classes of perfect graphs". Tech. rep., Humboldt-Universität zu Berlin. Manuscript. (Cited on page 90.)

Hsu J.C. (1996). *Multiple Comparisons. Theory and Methods.* Chapman & Hall. (Cited on pages 50, 52, 53, 55, 57, 65, 66, 68 and 132.)

Hurley S., Smith D.H., and Thiel S.U. (1997). "FASoft: A system for discrete channel frequency assignment". *Radio Science*, 32(5), pp. 1921–1939. (Cited on pages 167, 172, 177, 178, 183 and 188.)

Hutter F., Tompkins D.A.D., and Hoos H.H. (2002). "Scaling and probabilistic smoothing: Efficient dynamic local search for SAT". In *Principles and Practice of Constraint Programming – CP 2002*, edited by P.V. Hentenryck, vol. 2470 of *Lecture Notes in Computer Science*, pp. 233–248. Springer Verlag, Berlin, Germany. (Cited on pages 28 and 122.)

Ibaraki T., Imahori S., Kubo M., Masuda T., Uno T., and Yagiura M. (2004). "Effective local search algorithms for routing and scheduling problems with general time-window constraints". *Transportation Science*. To appear. (Cited on page 155.)

Igel C. and Toussaint M. (2003). "On classes of functions for which no free lunch results hold". *Information Processing Letters*. (Cited on page 33.)

Janssen J. and Kilakos K. (1999). "An optimal solution to the "philadelphia" channel assignment problem". *IEEE Transactions on Vehicular Technology*, 48(3), pp. 1012–1014. (Cited on page 174.)

Janssen J. and Narayanan L. (2001). "Approximation algorithms for the channel assignment problem". *Theoretical Computer Science A*, 262, pp. 649–667. Extended abstract published in the proceedings of ISAAC'99. (Cited on page 167.)

Janssen J. and Wentzell T. (2000). "Lower bounds from tile covers for the channel assignment problem". Tech. Rep. G-2000-09, GERAD, HEC, Montreal, Canada. (Cited on page 167.)

Jensen T.R. and Toft B. (1995). *Graph coloring problems*. Wiley Interscience, New York, USA. (Cited on page 167.)

Johnson D., Gutin G., McGeoch C., Yeo A., Zhang X., and Zverovitch A. (2002a). "Experimental analysis of heuristics for the ATSP". In Gutin and Punnen (2002), pp. 445–487. (Cited on page 5.)

Johnson D.S. (1974). "Worst-case behavior of graph-coloring algorithms". In *South-Eastern Conference on Combinatorics, Graph Theory and Computing*, edited by F. Hoffman, R. Kingsley, R. Levow, R. Mullin, and R. Thomas, vol. X of *Congressus Numerantium*, pp. 513–528. Utilitas Mathematica Publishing, Winnipeg, Canada. (Cited on page 89.)

Johnson D.S. (2002). "A theoretician's guide to the experimental analysis of algorithms". In Goldwasser et al. (2002), pp. 215–250. (Cited on page 39.)

Johnson D.S., Aragon C.R., McGeoch L.A., and Schevon C. (1991). "Optimization by simulated annealing: An experimental evaluation: Part II, graph coloring and number partitioning". *Operations Research*, 39(3), pp. 378–406. (Cited on pages 91, 94, 101, 103, 108, 123, 124, 143, 174, 232, 254, 287 and 290.)

Johnson D.S. and McGeoch L.A. (1997). "The Travelling Salesman Problem: A Case Study in Local Optimization". In Aarts and Lenstra (1997), pp. 215–310. (Cited on page 20.)

Johnson D.S. and McGeoch L.A. (2002). "Experimental analysis of heuristics for the STSP". In Gutin and Punnen (2002), pp. 369–443. (Cited on page 5.)

Johnson D.S., Mehrotra A., and Trick M. (eds.) (2002b). *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*. Ithaca, New York, USA. (Cited on pages 297, 301, 303, 305, 307, 309, 313 and 315.)

Johnson D.S., Papadimitriou C.H., and Yannakakis M. (1988). "How easy is local search?" *Journal of Computer and System Science*, 37(1), pp. 79–100. (Cited on page 34.)

Johnson D.S. and Trick M.A. (eds.) (1996). *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, 1993*, vol. 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, Providence, RI, USA. (Cited on pages 302, 312 and 314.)

Johri A. and Matula D. (1982). "Probabilistic bounds and heuristic algotihms for coloring large random graphs". Tech. Rep. 82-CSE-6, Southern Methodist University, Dallas, Texas, USA. (Cited on pages 89, 91, 97 and 124.)

Jones T. and Forrest S. (1995). "Fitness distance correlation as a measure of problem difficulty for genetic algorithms". In *Proceedings of the Sixth International Conference on Genetic Algorithms*, edited by L.J. Eshelman, pp. 184–192. Morgan Kaufmann Publishers, San Mateo, CA, USA. (Cited on page 78.)

Jonker R. and Volgenant A. (1987). "A shortest augmenting path algorithm for dense and sparse linear assignment problems". *Computing*, 38(4), pp. 325–340. (Cited on pages 123 and 240.)

Joslin D.E. and Clements D.P. (1999). "Squeaky wheel optimization". *Journal of Artificial Intelligence Research*, 10, pp. 353–373. (Cited on page 29.)

Karp R.M. (1972). "Reducibility among combinatorial problems". In *Complexity of computer computations*, edited by R.E. Miller and J.W. Thatcher, pp. 85–103. Plenum Press, New York, USA. (Cited on page 88.)

Kendall G. and Mohamad M. (2004). "Solving the fixed channel assignment problem in cellular communications using an adaptive local search". In Burke and Trick (2004), pp. 219–231. (Cited on page 167.)

Kernighan B. and Lin S. (1970). "An efficient heuristic procedure for partitioning graphs". *Bell System Technical Journal*, 49, pp. 291–307. (Cited on page 23.)

Khanna S., Motwani R., Sudan M., and Vazirani U. (1998). "On syntactic versus computational views of approximability". *SIAM Journal on Computing*, 28(1), pp. 164–191. (Cited on page 34.)

Kirkpatrick S., Jr. D.G., and Vecchi M.P. (1983). "Optimization by simulated annealing". *Science*, 220(4598), pp. 671–680. (Cited on page 24.)

Klotz W. (2002). "Graph coloring algorithms". Tech. Rep. Mathematik-Bericht 2002/5, Technische Universität Clausthal, Germany. (Cited on page 99.)

Knuth D. (1993). *The Stanford GraphBase: a platform for combinatorial computing*. ACM press, New York, NY, USA. (Cited on page 93.)

Kostuch P. (2003). "University course timetabling". Transfer Thesis, Oxford University, England. (Cited on page 233.)

Kostuch P. (2004). "The university course timetabling problem with a 3-phase approach". In Burke and Trick (2004), pp. 251–266. (Cited on pages 213, 233, 234 and 245.)

Kostuch P. and Socha K. (2004). "Hardness prediction for the university course timetabling problem". In *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2004*, edited by J. Gottlieb and G.R. Raidl, vol. 3004 of *Lecture Notes in Computer Science*, pp. 132–141. Springer Verlag, Berlin, Germany. (Cited on page 245.)

Lan K. and DeMets D. (1983). "Discrete sequential boundaries for clinical trials". *Biometrika*, 70(3), pp. 659–663. (Cited on page 68.)

Lanfear T.A. (1989). "Graph theory and radio frequency assignment". Tech. Rep. NATO Unclassified, Allied Radio Frequency Agency (ARFA), Brussels, Belgium. (Cited on page 172.)

Larrañaga P. and Lozano J.A. (2001). *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Kluwer Academic Publishers. (Cited on page 31.)

L'Ecuyer P. (1988). "Efficient and portable combined random number generators". *Communications of the ACM*, 31, pp. 742–749 and 774. (Cited on page 277.)

Lehmann E.L. (1986). *Testing statistical hypotesis*. John Wiley & Sons, New York, NY, USA, second ed. (Cited on page 61.)

Leighton F.T. (1979). "A graph coloring algorithm for large scheduling problems". *Journal of Research of the National Bureau of Standards*, 84(6), pp. 489–506. (Cited on pages 85 and 97.)

Lewandowski G. and Condon A. (1996). "Experiments with parallel graph coloring heuristics and applications of graph coloring". In Johnson and Trick (1996), pp. 309–334. (Cited on pages 85, 92 and 93.)

Lim A., Zhang X., and Zhu Y. (2003). "A hybrid method for the graph coloring and related problems". In *Proceedings of MIC'2003 – The Fifth Metaheuristics International Conference*. Kyoto-Japan. (Cited on pages 167, 168, 190, 191, 202 and 255.)

Lin S. and Kernighan B. (1973). "An effective heuristic algorithm for the travelling-salesman problem". *Operations Research*, 21(2), pp. 498–516. (Cited on pages 23, 111, 112 and 113.)

Loughlin T.M. and Noble W. (1997). "A permutation test for effects in an unreplicated factorial design". *Technometrics*, 39(2), pp. 180–190. (Cited on page 59.)

Lourenço H.R., Martin O., and Stützle T. (2002). "Iterated local search". In Glover and Kochenberger (2002b), pp. 321–353. (Cited on page 28.)

Lourenço H.R. (1995). "Job-shop scheduling: computational study of local search and large-step optimization methods". *European Journal of Operational Research*, 83(2). (Cited on page 33.)

Luczak T. (1991). "The chromatic number of random graphs." *Combinatorica*, 11(1), pp. 45–54. (Cited on page 89.)

Mahadev N. and Roberts F. (2003). "Consensus list colorings of graphs and physical mapping of DNA". In *Bioconsensus*, edited by M. Janowitz, F.J. Lapointe, F. McMorris, B. Mirkin, and F. Roberts, vol. 61 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pp. 83–95. American Mathematical Society, Providence, RI, USA, Providence, RI. (Cited on page 165.)

Maron O. and Moore A. (1994). "Hoeffding races: Accelerating model selection search for classification and function approximation". In *Advances in Neural Information Processing Systems*, edited by J.D. Cowan, G. Tesauro, and J. Alspector, vol. 6, pp. 59–66. Morgan Kaufmann Publishers, San Francisco, CA, USA. (Cited on page 67.)

Maron O. and Moore A.W. (1997). "The racing algorithm: Model selection for lazy learners." *Artificial Intelligence Review*, 11(1-5), pp. 193–225. (Cited on page 67.)

Marriott K. and Stuckey P.J. (1998). *Programming with Constraints: An Introduction*. MIT Press, Cambridge, Massachusetts. (Cited on page 18.)

Marx D. (2004). *Graph Coloring with Local and Global Constraints*. Ph.D. thesis, Budapest University of Technology and Economics, Hungary. (Cited on page 167.)

Mastrolilli M. and Gambardella L. (2004). "Maximum satisfiability: How good are tabu search and plateau moves in the worst-case?" *European Journal of Operational Research*, 166(1), pp. 63–76. (Cited on page 34.)

McAllester D., Selman B., and Kautz H. (1997). "Evidence for invariants in local search". In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pp. 321–326. AAAI Press / The MIT Press, Menlo Park, CA, USA. (Cited on page 120.)

McGeoch C., Sanders P., Fleischer R., Cohen P.R., and Precup D. (2002). "Using finite experiments to study asymptotic performance". In Fleischer et al. (2002), pp. 93–126. (Cited on pages 193 and 194.)

McGeoch C.C. (1992). "Analyzing algorithms by simulation: Variance reduction techniques and simulation speedups". *ACM Computing Surveys*, 24(2), pp. 195–212. (Cited on page 45.)

McRoberts K.L. (1971). "A search model for evaluating combinatorially explosive problems". *Operations Research*, 19(6), pp. 1331–1349. (Cited on pages 41 and 277.)

Mehrotra A. and Trick M. (1996). "A column generation approach for graph coloring". *INFORMS Journal On Computing*, 8(4), pp. 344–354. (Cited on pages 94, 172 and 254.)

Merlot L.T.G., Boland N., Hughes B.D., and Stuckey P.J. (2003). "A hybrid algorithm for the examination timetabling problem". In Burke and De Causmaecker (2003), pp. 207–231. (Cited on page 208.)

Merz P. (2000). *Memetic algorithms for combinatorial optimization problems: fitness landscapes and effective search strategies*. Ph.D. thesis, Department of Electrical Engineering and Computer Science, University of Siegen, Germany. (Cited on page 76.)

Merz P. and Freisleben B. (2000). "Fitness landscape analysis and memetic algorithms for the quadratic assignment problem". *IEEE Transactions on Evolutionary Computation*, 4(4), pp. 337–352. (Cited on page 76.)

Metropolis N., Rosenbluth A., Rosenbluth M., Teller A., and Teller E. (1953). "Equation of state calculation by fast computing machines". *Journal of Chemical Physics*, 21, pp. 1087–1092. (Cited on page 25.)

Michalewicz Z. and Fogel D.B. (2000). *How to solve it: Modern Heuristics*. Springer Verlag, Berlin, Germany. (Cited on page 206.)

Mills P. and Tsang E. (2000). "Guided local search for solving SAT and weighted MAX-SAT problems". *Journal of Automated Reasoning*, 24(1-2), pp. 205–223. Special Issue on Satisfiability Problems. (Cited on pages 28 and 122.)

Minton S., Johnston M., Philips A., and Laird P. (1992). "Minimizing conflicts: A heuristic repair method for constraint satisfaction and scheduling problems". *Artificial Intelligence*, 58(1-3), pp. 161–205. (Cited on page 120.)

Mizuno K. and Nishihara S. (2002). "Toward ordered generation of exceptionally hard instances for graph 3-colorability". In Johnson et al. (2002b), pp. 1–8. (Cited on page 93.)

Monasson R., Zecchina R., Kirkpatrick S., Selman B., and Troyansky L. (1999). "Computational complexity from "characteristic" phase transitions". *Nature*, 400, pp. 133–137. (Cited on page 80.)

Montemanni R. (2001). *Upper and lower bounds for the fixed spectrum frequency assignment problem*. Ph.D. thesis, Division of Mathematics and Statistics, School of Technology, University of Glamorgan, UK. (Cited on page 167.)

Montgomery D.C. (2000). *Design and Analysis of Experiments*. John Wiley & Sons, fifth ed. (Cited on pages 47, 51, 52 and 278.)

Moret B. (2002). "Towards a discipline of experimental algorithmics". In Goldwasser et al. (2002), pp. 197–213. (Cited on pages 38 and 39.)

Morgenstern C. (1996). "Distributed coloration neighborhood search". In Johnson and Trick (1996), pp. 335–357. (Cited on page 106.)

Morgenstern C. and Shapiro H. (1990). "Coloration neighborhood structures for general graph coloring". In *Proceedings of the first annual ACM-SIAM Symposium on Discrete algorithms*, pp. 226–235. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA. (Cited on pages 103 and 106.)

Motwani R. and Raghavan P. (1995). *Randomized Algorithms*. Cambridge University Press, Cambridge, UK. (Cited on page 19.)

Murphey R.A., Pardalos P.M., and Resende M.G.C. (1999). "Frequency assignment problems". In *Handbook of combinatorial optimization*, edited by D.Z. Du and P.M. Pardalos, vol. A, Supplement. kap. (Cited on pages 166 and 176.)

Naudts B. and Kallel L. (2000). "A comparison of predictive measures of problem difficulty in evolutionary algorithms". *IEEE Transactions on Evolutionary Computation*, 4(1), pp. 1–16. (Cited on page 79.)

Newall J.P. (1999). *Hybrid methods for automated timetabling*. Ph.D. thesis, Department of Computer Science, University of Nottingham, UK. (Cited on page 219.)

Ovacik I.M., Rajagopalan S., and Uzsoy R. (2000). "Integrating interval estimates of global optima and local search methods for combinatorial optimization problems". *Journal of Heuristics*, 6(4), pp. 481–500. (Cited on page 41.)

Papadimitriou C.H. (1994). *Computational Complexity*. Addison-Wesley, Reading, MA, USA. (Cited on page 14.)

Papadimitriou C.H. and Steiglitz K. (1982). *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., Englewood Cliffs, NJ. (Cited on pages 4, 14 and 225.)

Paquete L. (2005). *Stochastic Local Search Algorithms for Multiobjective Combinatorial Optimization: Methods and Analysis*. Ph.D. thesis, Technische Universität Darmstadt, Darmstadt, Germany. Forthcoming. (Cited on page 70.)

Paquete L., Chiarandini M., and Stützle T. (2004). "Pareto local optimum sets in the biobjective traveling salesman problem: An experimental study". In *Metaheuristics for Multiobjective Optimisation*, edited by X. Gandibleux, M. Sevaux, K. Sörensen, and V. T'kindt, vol. 535 of *Lecture Notes in Economics and Mathematical Systems*. Springer Verlag, Berlin, Germany. (Cited on page 79.)

Paquete L. and Fonseca C. (2001). "A study of examination timetabling with multiobjective evolutionary algorithms". In *Proceedings of MIC'2001 – Meta–heuristics International Conference*, pp. 149–154. Porto, Portugal. (Cited on page 209.)

Paquete L. and Stützle T. (2002a). "Empirical analysis of tabu search for the lexicographic optimization of the examination timetabling problem". In *Proceedings of the Practice and*

Theory of Automated Timetabling (PATAT 2002), edited by E. Burke and P. Causmaecker, pp. 413–420. Also available as Technical Report AIDA-02-02, FG Intellektik, FB Informatik, Technische Universität Darmstadt, 2002. (Cited on page 209.)

Paquete L. and Stützle T. (2002b). "An experimental investigation of iterated local search for graph coloring". In *Applications of Evolutionary Computing*, edited by S. Cagnoni, J. Gottlieb, E. Hart, M. Middendorf, and G. Raidl, vol. 2270 of *Lecture Notes in Computer Science*, pp. 122–131. Springer Verlag, Berlin, Germany. (Cited on page 122.)

Pardalos P. and Romeijn E. (eds.) (2002). *Handbook of Global Optimization, Volume 2: Heuristic Approaches*. Kluwer Academic Publishers. (Cited on page 39.)

Paschos V.T. (2003). "Polynomial approximation and graph-coloring". *Computing*, 70(1), pp. 41–86. (Cited on page 89.)

Peemöller J. (1983). "A correction to Brelaz's modification of Brown's coloring algorithm". *Communications of the ACM archive*, 26(8), pp. 595–597. (Cited on page 94.)

Pelikan M., Goldberg D.E., and Cantú-Paz E. (1999). "BOA: The Bayesian optimization algorithm". In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, edited by W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, and R.E. Smith, vol. I, pp. 525–532. Morgan Kaufmann Publishers, San Francisco, CA, USA. (Cited on page 31.)

Pesant G. and Gendreau M. (1996). "A view of local search in constraint programming". In *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming*, edited by E.C. Freuder, vol. 1118 of *Lecture Notes in Computer Science*, pp. 353–366. Springer. (Cited on page 32.)

Pesarin F. (2001). *Multivariate Permutation Tests. With applications in Biostatistics.* John Wiley & Sons, New York, NY, USA. (Cited on pages 48, 49, 54, 59, 60, 63 and 73.)

Phan V. and Skiena S. (2002). "Coloring graphs with a general heuristic search engine". In Johnson et al. (2002b), pp. 92–99. (Cited on page 167.)

Popper K. (1963). *Conjectures and Refutations*. Routledge. (Cited on page 3.)

Post G. and Veltman B. (2004). "Harmonious personnel scheduling". In Burke and Trick (2004), pp. 557–559. (Cited on page 208.)

Press W.H., Teukolsky S.A., Vetterling W.T., and Flannery B.P. (1992). *Numerical Recipes in C: the art of scientific computing*. Cambridge University Press, Cambridge, UK, second ed. (Cited on pages 57 and 277.)

Prestwich S. (2002a). "Coloration neighbourhood search with forward checking". *Annals of Mathematics and Artificial Intelligence*, 34(4), pp. 327–340. (Cited on page 137.)

Prestwich S. (2002b). "Constrained bandwidth multicoloration neighbourhoods". In Johnson et al. (2002b), pp. 126–133. (Cited on pages 33, 188, 191 and 255.)

Prestwich S. (2003). "Hybrid local search on two multicolouring models". In *International Symposium on Mathematical Programming*. Copenhagen, Denmark. (Cited on pages 167, 188, 191 and 197.)

Rardin R.L. and Uzsoy R. (2001). "Experimental evaluation of heuristic optimization algorithms: A tutorial". *Journal of Heuristics*, 7(3), pp. 261–304. (Cited on pages 5, 39, 40 and 45.)

Raychaudhuri A. (1994). "Further results on T-coloring and frequency assignment problems". *SIAM Journal on Discrete Mathematics*, 7(4), pp. 605–613. (Cited on page 173.)

Risler M., Chiarandini M., Paquete L., Schiavinotto T., and Stützle T. (2004). "An algorithm for the car sequencing problem of the ROADEF 2005 challenge." Tech. Rep. AIDA–04–06, FG Intellektik, FB Informatik, Technische Universität Darmstadt, Darmstadt, Germany. (Cited on pages 35 and 246.)

Roberts F.S. (1991). "T-colorings of graphs: Recent results and open problems". *Discrete Mathematics*, 93(2-3), pp. 229–245. (Cited on pages 166 and 167.)

Robertson N., Sanders D.P., Seymour P., and Thomas R. (1996). "A new proof of the four-colour theorem". *Electronic Research Announcements of the American Mathematical Society*, 2(1), pp. 17–25. (Cited on page 88.)

Rochat Y. and Taillard E.D. (1995). "Probabilistic diversification and intensification in local search for vehicle routing". *Journal of Heuristics*, 1, pp. 147–167. (Cited on page 29.)

Rossi-Doria O. and Paechter B. (2003). "An hyperheuristic approach to course timetabling problem using evolutionary algorithm". Tech. Rep. CC-00970503, Napier University, Edinburgh, Scotland. (Cited on page 226.)

Rossi-Doria O., Paechter B., Blum C., Socha K., and Samples M. (2003a). "A local search for the timetabling problem". In Burke and De Causmaecker (2003), pp. 124–127. (Cited on pages 219, 222 and 225.)

Rossi-Doria O., Samples M., Birattari M., Chiarandini M., Knowles J., Manfrin M., Mastrolilli M., Paquete L., Paechter B., and Stützle T. (2003b). "A comparison of the performance of different metaheuristics on the timetabling problem". In Burke and De Causmaecker (2003), pp. 329–351. (Cited on page 219.)

Rudolph G. (1994). "Convergence analysis of canonical genetic algorithms". *IEEE Transactions on Neural Networks*, 5(1), pp. 96–101. (Cited on page 33.)

Russell S. and Norvig P. (2003). *Artificial Intelligence: A Modern Approach*. Prentice Hall, Englewood Cliffs, New Jersey, USA. Second. (Cited on pages 4 and 17.)

Savelsbergh M., Uma R., and Wein J. (2005). "An experimental study of LP-based approximation algorithms for scheduling problems". *INFORMS Journal on Computing*, 17(1), pp. 123–136. (Cited on page 39.)

Schiavinotto T. and Stützle T. (2004). "The linear ordering problem: Instances, search space analysis and algorithms". *Journal of Mathematical Modelling and Algorithms*, 3(4), pp. 367–402. (Cited on page 76.)

Schiavinotto T. and Stützle T. (2005). "Metrics on permutations for search landscape analysis". In preparation. (Cited on page 79.)

Schindl D. (2003). "Graph coloring and linear programming". Presentation at First Joint Operations Research Days, Ecole Polytechnique Fédérale de Lausanne (EPFL), available on line at `http://roso.epfl.ch/ibm/jord03.html`. (June 2005). (Cited on page 94.)

Schreiber G.R. and Martin O.C. (1999). "Cut size statistics of graph bisection heuristics". *SIAM Journal on Optimization*, 10(1), pp. 231–251. (Cited on page 20.)

Schumacher C., Vose M.D., and Whitley L.D. (2001). "The no free lunch and problem description length". In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, edited by L. Spector, E.D. Goodman, A. Wu, W.B. Langdon, H.M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M.H. Garzon, and E. Burke, pp. 565–570. Morgan Kaufmann Publishers, San Francisco, CA, USA. (Cited on page 33.)

Schuurman P. and Woeginger G.J. (2001). "Approximation schemes – a tutorial". Tech. Rep. Report Woe-65, Technische Universität Graz, Graz, Austria. (Cited on page 14.)

Sedgewick R. (1988). *Algorithms*. Addison-Wesley, Reading, MA, USA, second ed. (Cited on page 225.)

Setubal J.C. (1996). "Sequential and parallel experimental results with bipartite matching algorithms". Tech. Rep. EC-96-09, Institute of Computing, University of Campinas, Brasil. (Cited on pages 5 and 226.)

Shaffer J.P. (1995). "Multiple hypothesis testing". *Annual Review of Psychology*, 46, pp. 561–576. (Cited on page 50.)

Sheskin D.J. (2000). *Handbook of Parametric and Nonparametric statistical procedures*. Chapman & Hall, second ed. (Cited on pages 48, 51, 55, 56, 57, 58 and 65.)

Silva J.D.L., Burke E.K., and Petrovic S. (2004). "An introduction to multiobjective metaheuristics for scheduling and timetabling". In *Metaheuristic for Multiobjective Optimisation*, edited by X. Gandibleux, M. Sevaux, K.Sorensen, and V. T'kindt, vol. 535 of *Lecture Notes in Economics and Mathematical Systems*, pp. 91–129. Springer Verlag, Berlin, Germany. (Cited on page 209.)

Simon H.U. (1989). "Approximation algorithms for channel assignment in cellular radio networks". In *Fundamentals of computation theory*, vol. 380 of *Lecture Notes in Computer Science*, pp. 405–415. Springer Verlag, Berlin, Germany. (Cited on page 167.)

Sivarajan K.N., McEliece R.J., and Ketchum J.W. (1989). "Channel assignment in cellular radio". In *Proceedings of the 39th IEEE Vehicular Technology Conference*, pp. 846–850. (Cited on page 180.)

Skiena S. and Berend D. (2004). "Combinatorial dominance and heuristic search". Manuscript. (Cited on page 274.)

Smith B.C. and Sucur M. (1996). "Analysis of solution quality in scheduling planning". In *The 36th Annual Symposium of AGIFORS*. Atlanta, Georgia, USA. (Cited on pages 41 and 277.)

Smith D.H., Allen S.M., and Hurley S. (2002). "Characteristics of good meta-heuristic algorihtms for the frequency assignment problem". *Annals of Operations Research*, 107(1–4), pp. 285–301. (Cited on page 167.)

Smith D.H., Hurley S., and Allen S.M. (2000). "A new lower bound for the channel assignment problem". *IEEE Transactions on Vehicular Technology*, 49(4), pp. 1265–1272. (Cited on page 167.)

Socha K. (2003). "The influence of run-time limits on choosing ant system parameters". In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, edited by Cantu-Paz et al., vol. 2723 of *Lecture Notes in Computer Science*, pp. 49–60. Springer Verlag, Berlin, Germany. (Cited on page 220.)

Socha K., Sampels M., and Manfrin M. (2003). "Ant algorithms for the university course timetabling problem with regard to the state-of-the-art". In *Proceedings of EvoCOP 2003 – 3rd European Workshop on Evolutionary Computation in Combinatorial Optimization*, edited by S. Cagnoni, J.R. Cardalda, D. Corne, J. Gottlieb, A. Guillot, E. Hart, C. Johnson, E. Marchiori, J.A. Meyer, M. Middendorf, and G. Raidl, vol. 2611 of *Lecture Notes in Computer Science*, pp. 334–345. Springer Verlag, Berlin, Germany. (Cited on page 220.)

Stadler P.F. (1996). "Landscapes and their correlation functions". *Journal of Mathematical Chemistry*, 20(1), pp. 1–45. (Cited on page 78.)

Stockmeyer L. (1973). "Planar 3-colorability is NP-complete". *SIGACT News*, 5(3), pp. 19–25. (Cited on page 88.)

Stützle T. (1998). *Local Search Algorithms for Combinatorial Problems — Analysis, Improvements, and New Applications*. Ph.D. thesis, FB Informatik, Technische Universität Darmstadt, Darmstadt, Germany. (Cited on pages 24, 30, 120 and 253.)

Stützle T. and Dorigo M. (2002). "A short convergence proof for a class of ACO algorithms". *IEEE Transactions on Evolutionary Computation*, 6(4), pp. 358–365. (Cited on page 33.)

Stützle T. and Hoos H.H. (2000). "$\mathcal{MAX}$–$\mathcal{MIN}$ Ant System". *Future Generation Computer Systems*, 16(8), pp. 889–914. (Cited on page 30.)

Stützle T. and Hoos H.H. (2001). "Analysing the run-time behaviour of iterated local search for the travelling salesman problem". In *Essays and Surveys on Metaheuristics*, edited by P. Hansen and C.C. Ribeiro, pp. 589–611. Kluwer Academic Publishers, Boston, MA, USA. (Cited on page 75.)

Taillard E. (2001). "Comparaison of non-deterministic iterative methods". In *Proceedings of MIC'2001 – Meta–heuristics International Conference*, pp. 273–276. Porto, Portugal. (Cited on page 72.)

Taillard E.D. (1991). "Robust taboo search for the quadratic assignment problem." *Parallel Computing*, 17(4-5), pp. 443–455. (Cited on page 26.)

Talbi E.G. (2002). "A taxonomy of hybrid metaheuristics". *Journal of Heuristics*, 8(5), pp. 541–564. (Cited on page 31.)

ter Braak C.J. (1992). "Permutation versus bootstrap significance tests in multiple regression and ANOVA". In *Bootstrapping and related techniques*, edited by K.H. Jöckel, G. Rothe, and W. Sendler, pp. 79–86. Springer-Verlag, Berlin. (Cited on page 275.)

Terashima-Marín H., Ross P., and Valenzuela-Rendón. M. (1999). "Evolution of constraint satisfaction strategies in examination timetabling". In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, edited by W. Banzhaf, J.M. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M.J. Jakiela, and R.E. Smith, pp. 635–642. Morgan Kaufmann Publishers, San Francisco, CA, USA. (Cited on page 227.)

Tesman B.A. (1990). "Set T-colorings". *Congrussus Numerantium*, 77, pp. 229–242. (Cited on pages 163 and 167.)

Tesman B.A. (1993). "List T-colorings". *Discrete Appied Mathematics*, 45(3), pp. 277–289. (Cited on pages 163 and 167.)

Thompson J. and Dowsland K. (1998). "A robust simulated annealing based examination timetabling system". *Computers and Operations Research*, 25(7-8), pp. 637–648. (Cited on pages 209 and 228.)

Thompson P. and Orlin J. (1989). "The theory of cycle transfers". Tech. Rep. 200-89, MIT Operations Research Center, Cambridge, MA. (Cited on pages 35, 107 and 155.)

Tsang E. and Voudouris C. (1998). "Solving the radio link frequency assignment problem using guided local search". In *NATO Symposium on Radio Length Frequency Assignment*. Aalborg, Denmark. Also available on line at `http://cswww.essex.ac.uk/CSP/papers.html`. (June 2005). (Cited on page 188.)

Tuza Z. (1997). "Graph colorings with local restrictions – A survey". *Discussiones Mathematicae, Graph Theory*. (Cited on page 167.)

van der Tweel I. (2004). *Application and efficiency of sequential tests in matched case-control studies*. Ph.D. thesis, University Utrecht. (Cited on page 67.)

Vasquez M. (2004). "New results on the queens_n^2 graph coloring problem". *Journal of Heuristics*, 10(4), pp. 407–413. (Cited on page 96.)

Vazirani V.V. (2001). *Approximation Algorithms*. Springer-Verlag, Berlin. (Cited on page 14.)

Černý V. (1985). "Thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm." *Journal Of Optimization Theory And Applications*, 45(1), pp. 41–51. (Cited on page 24.)

Venables W.N. and Ripley B.D. (2002). *Modern Applied Statistics with S*. Springer-Verlag, Berlin, fourth ed. (Cited on pages 75, 131 and 150.)

Voudouris C., Dorne R., Lesaint D., and Liret A. (2001). "iOpt: A software toolkit for heuristic search methods." In *Proceedings of Principles and Practice of Constraint Programming - CP 2001*, edited by T. Walsh, vol. 2239 of *Lecture Notes in Computer Science*, pp. 716–719. Springer Verlag, Berlin, Germany. (Cited on page 208.)

Wald A. (1947). *Sequential Analysis*. John Wiley & Sons, New York, NY, USA. (Cited on page 67.)

Watson J.P. (2003). *Empirical modeling and anlysis of local search algorithms for the Job-Shop Scheduling Problem*. Ph.D. thesis, Department of Computer Science, Colorado State University, Fort Collins, CO. (Cited on page 76.)

Watson J.P., Beck J.C., Howe A.E., and Whitley L.D. (2003). "Problem difficulty for tabu search in job-shop scheduling". *Artificial Intelligence*, 143(2), pp. 189–217. Preprint. (Cited on pages 76, 79, 80, 238 and 241.)

Weihe K. (2001). "A software engineering perspective on algorithmics." *ACM Computing Surveys*, 33(1), pp. 89–134. (Cited on page 218.)

Weinberger E.D. (1990). "Correlated and uncorrelated fitness landscapes and how to tell the difference". *Biological Cybernetics*, 63(5), pp. 325–336. (Cited on page 78.)

White G.M. and Zhang J. (1998). "Generating complete university timetables by combining tabu search with constraint logic". In Burke and Carter (1998), pp. 187–200. (Cited on page 208.)

Wigderson A. (1983). "Improving the performance guarantee for approximate graph coloring". *Journal of the ACM*, 30(4), pp. 729–735. (Cited on page 89.)

Williams R., Gomes C.P., and Selman B. (2003). "Backdoors to typical case complexity." In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, IJCAI-03*, edited by G. Gottlob and T. Walsh, pp. 1173–1178. Morgan Kaufmann Publishers, CA, USA. (Cited on page 258.)

Wolpert D.H. and Macready W.G. (1997). "No free lunch theorems for optimization". *IEEE Transactions on Evolutionary Computation*, 1(1), pp. 67–82. (Cited on page 33.)

Wolsey L. (1998). *Integer Programming*. John Wiley & Sons. (Cited on page 16.)

Wright S. (1932). "The roles of mutation, inbreeding, crossbreeding and selection in evolution". In *International Proceedings of the Sixth International Congress on Genetics*, edited by D.F. Jones, vol. 1, pp. 356–366. (Cited on page 76.)

Yuan B. and Gallagher M. (2004). "Statistical racing techniques for improved empirical evaluation of evolutionary algorithms". In *Parallel Problem Solving from Nature - PPSN VIII, 8th International Conference, Birmingham, UK, September 18-22, 2004, Proceedings*, edited by Xin Yao et al., vol. 3242 of *Lecture Notes in Computer Science*, pp. 172–181. Springer Verlag, Berlin, Germany. (Cited on page 67.)

Zemel E. (1981). "Measuring the quality of approximate solutions to zero-one programming problems". *Mathematics of operations research*, 6(3), pp. 319–332. (Cited on page 41.)

Zervoudakis K. and Stamatopoulos P. (2001). "A generic object-oriented constraint-based model for university course timetabling". In Burke and Erben (2001), pp. 28–47. (Cited on page 208.)

Zlochin M. and Dorigo M. (2002). "Model-based search for combinatorial optimization: A comparative study". In *Parallel Problem Solving from Nature - PPSN VII, 7th International Conference*, edited by Merelo Guervós J.J. et al., vol. 2439 of *Lecture Notes in Computer Science*, pp. 651–661. Springer Verlag, Berlin, Germany. (Cited on page 41.)

Zymolka A., Koster A.M.C.A., and Wessäly R. (2003). "Transparent optical network design with sparse wavelength conversion". In *Proceedings of ONDM 2003, the 7th IFIP Working Conference on Optical Network Design & Modelling*, pp. 61–80. Budapest, Hungary. (Cited on pages 85 and 92.)

"[...] si quis in caelum ascendisset
naturamque mundi et pulcheritudinem siderum perspexisset,
insuavem illam admirationem ei fore,
quae iucundissima fuisset, si aliquem, cui narraret, habuisset."

M. Tulli Ciceronis, "Laelius de Amicitia", XXIII, 88.